Contract No. DA-36-034-ORD-1646

FINAL REPORT

on

CONTRACT NO. DA-36-034-ORD-1646

PART II (COMPUTER USE)

for the period from

1 July 1954 to 31 December 1956

THE INSTITUTE FOR ADVANCED STUDY
ELECTRONIC COMPUTER PROJECT
PRINCETON, NEW JERSEY

Contract No. DA-36-034-ORD-1646
Project No. TB3-0538

FINAL REPORT

on

Contract No. DA-36-034-ORD-1646

PART II (COMPUTER USE)

by

The Staff
Electronic Computer Project

THE INSTITUTE FOR ADVANCED STUDY
ELECTRONIC COMPUTER PROJECT
PRINCETON, NEW JERSEY

May 1957

TABLE OF CONTENTS

PREFACE

The following report has been prepared in accordance with the terms of Contract No. DA-36-034-ORD-1646 and constitutes the Final Report called for under the terms of that contract.

Said Contract between the Institute for Advanced Study and the Department of the Army was entered on June 2, 1954, for "further development of principals and methods for operation and maintenance of very high speed digital electric computer devices", i.e. for continuation of our work under Contract No. DA-36-034-ORD-1330, which terminated on June 30, 1954, and for which a final report was submitted, dated December 1954.

Contract No. DA-36-034-ORD-1646, together with its supplemental agreements No. 1 and No. 2, was in effect from July 1, 1954 through October 31, 1956 and was the only contract supporting maintenance and operation of our Computer during that period, except for the last 7 months during which a small part of the operation has been paid for under Contract Nonr 1358-(04). This new contract between the Institute for Advanced Study and the Office of Naval Research has taken over the full burden of machine maintenance and operation on January 1, 1957.

This Final Report is divided into two parts:

PART I covers the engineering work carried out from July 1, 1954 through December 31, 1956 under the terms of Contract No DA-36-034-ORD-1646.

PART II lists a number of problems for which, during the same 30 months, numerical results have been obtained with the help of our Computer. This work was supported by four contracts:

(1) Contract DA-36-034-ORD-1646 and Nonr 1358-(04) for machine operation.

(2) Contract N7-onr-388 for the development of methods for high-speed automatic computing (from July 1, 1954 through December 31, 1954.)

(3) Contract Nonr 1358-(03) for the development of methods for high-speed automatic computing (since January 1, 1955).

(4) Contract Nonr 1358-(02) for all mathematical and coding work connected with meteorological research.

During the whole period our Computer was used for scientific computations exclusively and no charge was made to any other organization or contract for coding or machine operation.

Hans J. Maehly

Hans J. Maehly
Acting Project Director

ACKNOWLEDGEMENTS

The late Prof. J. von Neumann, originator and director of our project, left Princeton soon after the beginning of the period covered by this report. Dr. H. H. Goldstine, who has been Acting Director until 30 June 1956, has contributed to most of the problems described hereinafter. The names of the individuals working on the various problems are given under the respective headings.

I. GENERAL PURPOSE ROUTINES

10. General Remarks

It was decided early in 1956 that no new machine should be built at the Institute for Advanced Study; that most of the engineering staff would, therefore, leave to pursue development work at other places; and that the Electronic Computer should be transformed from an experimental project into a tool for the solution of the many computational problems arising in the scientific community of Princeton. This transformation will be no minor task, as no attempt has been made, during the construction and completion of our equipment, to provide for easy communication between the computing machine and the coder or mathematician; and it is made even more difficult by the requirement that, lacking funds for a staff of professional coders, coding procedures should be simplified so that most problems can be coded by their originators.

After 1 July 1956, the development of a consistent system of general purpose routines has, therefore, been given priority over all new work, which simply means that no major help could be rendered to anybody presenting problems after this date. We felt obliged, however, to finish those problems whose coding was already under way. This, together with the engineering changes and difficulties described in volume I of this report, has delayed the development of our general purpose routines much more than originally anticipated.

Fortunately, the floating-point interpretive routine (FLINT) went already into operation late in 1955 and, despite its admitted drawbacks, has largely filled the gap between the restriction of our coding services and the completion of other service routines. Besides its operational applica-

tions, the modified interpreter (chapter 11) has yielded valuable suggestions for an improved version which is still being planned. At this point, however, emphasis is being put on the construction of subroutines, up to a symbolic assembler*), to facilitate coding in direct, fixed-point machine language (chapter 13).

Our staff has been too small to engage in any ambitious projects involving new concepts. What is now most needed, are standard utility routines, which, unfortunately, we cannot copy from any other place since no machine resembles ours closely enough. A description of our new service routines is included in this report as some of the details may be new, or of interest where the construction of similar codes or machines is being contemplated.

---

*) To be described in Final Report for Contract Nonr-1358-(04).

11. FLINT = FLOATING POINT INTERPRETIVE SUBROUTINE

Originator and Coder: Hans J. Maehly

11.0 Purpose and Applications

During the construction of the IAS Computer, emphasis was on speed and flexibility rather than on ease of coding. It was thus possible to carry out problems which had been too long or too unusual in their kind for earlier computers. It must be admitted, on the other hand, that routine problems for which many other computers would have been quite adequate took an undue amount of coding time, while the saving in running time (e.g. 2 rather than 10 minutes) was insignificant. Such small problems did arise in the Princeton scientific community and it would have been impractical not to solve them here as long as computer time was available.

An auxiliary routine has therefore been written which, as far as its user is concerned, transforms our machine into a slower, less sophisticated instrument for which coding is much simpler. Meanwhile, experience has shown that the effort to write this routine (approximately one half man-year) was a well worth-while investment. FLINT has been used not only by an increasing number of occasional users to code their own problems but also by the permanent staff of our organization for several auxiliary investigations and even for parts of large problems.

11.1 Basic Structure

The basic features of FLINT are indicated by its name: floating-point interpretive subroutine.

Each floating-point number is stored in one word = 40 binary bits; the first 31 bits are reserved for the "mantissa" r and the last 9 bits for the "exponent" ρ . Their respective ranges are:

$$\frac{1}{2} \leq |r| < 1$$
$$-256 < \rho < 256 \qquad *)$$

*) Actually $256+\rho$ is stored in the last 9 bits in order to prevent carry into the mantissa part.

In all operations (such as +, -, x, $\frac{\cdot}{\cdot}$) this word is treated as a number x

$$x = r.2^{\varphi} ,$$

hence $2^{-256} \leq |x| < 2^{+255}$

or appr. $10^{-77} \leq |x| < 5.10^{76}$

This range is wide enough to avoid scaling problems for all but the most extreme cases (such as combinatorial problems with more than 58! or some astrophysical computations). The remaining 31 binary digits for the mantissa still yield a relative accuracy of 9 decimal digits.

An interpretive routine is, by definition, a code that "translates" orders given in a new "language" into ordinary "machine language". Thus, every FLINT order is picked up by FLINT as soon as the previous order has been executed, is then interpreted and causes a series of machine operations which provide the changes required by the given FLINT order. For example: a FLINT multiplication order will cause FLINT to multiply the mantissae and to add the exponents of the two respective floating-point numbers. Thus the machine plus FLINT will act like a new machine though no physical changes have been made for that purpose. We shall, therefore, speak of FLINT as if it were a virtual machine rather than an auxiliary code.

The length of an order in FLINT has been cut down to 10 bits (as compared with 20 for "direct coding"), which are divided into two pentades--00000 through 11111, i.e. 0 through 31. The first pentade defines the operation, the second one gives the address of the operand. Each code written for the use with FLINT must be segmented into blocks of less than 30 words (120 orders); all blocks will be transferred to the magnetic drum during the read-in and only one at a time will, for execution, be called into the Williams Memory where line 30 has been set aside for this purpose. These blocks are, therefore, often referred to as "lines" of FLINT code.

Another transfer from the drum is necessary to invoke either of the two subroutines which provide for the conversion of floating point decimal numbers, punched on and read in from IBM cards, to floating point binary

form; or for the reconversion from binary to decimal for punching out.

All of these operations -- transfer to another line of FLINT code, conversion, and reconversion -- are initiated by a single order and include automatic checking and, for the latter two orders, the necessary IBM operations (read-in / punch-out).

As an aid in finding errors in one's program, FLINT is available in a Tracing mode in which the contents of the X and A accumulators and the Y register, defined in the following paragraphs, are punched out, together with the address of the instruction word to which control is about to be transferred. This punching takes place at the end of every instruction word, i.e. after about every four orders. A more flexible Tracer is being prepared, but the current one has been found to be quite valuable.

11.2 "Direct" Floating Arithmetic Orders

FLINT is a single-address system. Most of the arithmetic that FLINT performs on the customer's data will be done in an accumulator which we refer to by the letter X. When not being processed by the accumulator, data are stored in floating-point registers, m. The contents of any register are denoted by ( ), and in the case of the accumulator X, they may also be referred to as x. The fundamental arithmetic orders are Read (into the accumulator), Add, Subtract, Multiply, Divide, and Store (in floating-point storage registers). The orders which cause these operations are shown below:

(10,m)	(m) $\longrightarrow$ X
(11,m)	x + (m) $\longrightarrow$ X
(12,m)	x - (m) $\longrightarrow$ X
(13,m)	x . (m) $\longrightarrow$ X
(14,m)	x / (m) $\longrightarrow$ X
(15,m)	x $\longrightarrow$ m, non-clear X

In each of these orders we may refer by the number m to any of some thirty data registers. (The binary character of the IAS machine shows through in the FLINT code by the great importance attached to the number 32. The order codes may take on the values from 0 to 31. Groups of addresses are treated systematically except possibly for the numbers at each end of the sequence,

i.e. the addresses 0 to 31, and occasionally 30.)

As an example of these orders, the order (11,17) would cause the floating-point number in memory register 17 to be added to whatever was currently standing in X and, of course, it would not disturb the contents of register 17. Similarly, the order (15,14) would cause the contents of the accumulator X to be stored in register 14 and would not disturb the contents of the accumulator.

Although the arithmetic orders already given will suffice to carry out any operations that may be needed, it is convenient to have a few more. The orders (16,m) through (19,m) cause FLINT to execute the instructions shown below:

(16,m)	(m)	$\longrightarrow$ Y
(17,m)	x + (m).y	$\longrightarrow$ X
(18,m)	x - (m).y	$\longrightarrow$ X
(19,m)	(m) / x	$\longrightarrow$ X

The order (16,m) is a "Fetch" order that brings a floating-point number from register m into a special register designated by the letter Y whose contents we denote -- in analogy to the X accumulator -- by y. This register is, in fact, register 0 -- so the same result could be obtained by the order (10,m) followed by (15,00); however, such an order pair not only requires twice as many FLINT instructions but it also destroys the contents of our regular X accumulator. The orders (17,m) and (18,m) invoke both the X accumulator and the Y register, causing the product of the floating-point number in address m by the floating-point number in the Y register to be added to or subtracted from the contents of the X accumulator, leaving the result in the X accumulator. This set of orders permits the expedient calculation of the dot product of two vectors -- an extremely common form of numerical manipulation. - The (19,m) order is an inverse division which permits the contents of floating register m to be divided by the contents of X. Experience has shown that this order is used at least as frequently as the normal division (14,m). It thus seems very desirable to have an inverse division in future machines.

11.3 Integer Arithmetic Orders

In most digital computing, expediency dictates that the basic program be written in an iterative fashion. That is, the same set of orders is performed over and over again, either identically or with slight modifications of the address parts of the orders, thereby performing the same operation systematically on large sets of similar data. Both in going around simple loops and in applying the same set of orders to large sets of data it is necessary to count how many times we have been around and to modify addresses. For all these purposes, it is necessary to do simple arithmetic with integers. In the IAS machine integers may be interpreted as addresses in the high-speed memory if they lie in the range from 0 to 1023. In order to carry out integer arithmetic, a second arithmetic organ is provided by FLINT. The accumulator of the integer arithmetic organ is designated by A and its contents, in accordance with previous usage, by a. We have a set of instructions applying to the A accumulator, which is quite similar to our floating arithmetic orders. We can Fetch numbers into A, Add, Subtract, Multiply and Store the results of A back in storage registers. The detailed order code is given below.

(00,n)	(n) $\longrightarrow$ A
(01.n)	a + (n) $\longrightarrow$ A
(02,n)	a - (n) $\longrightarrow$ A
(03,n)	a . (n) $\longrightarrow$ A
(04,n)	a $\neq$ (n) ? *)
(05,n)	a $\longrightarrow$(n), non-clear A

We have again some thirty storage registers but they are not the same storage registers which were referred to by the floating-point arithmetic orders. These storage registers are used only for storing integers in the range from 0 to 1023 (that is, integers representable by 10 binary digits).

The A accumulator is identical with the integer register whose address is 0. (This in contradistinction to the X accumulator, which is not identical with any of the addressable floating-point storage registers). The choice of integer register 0 for the A accumulator has been intentional

---

*) explanation on next page.

so that the order (00,00) should be a no-operation order -- since it fetches the contents of integer register 0 to itself.

The operation of division is missing. If we divide two integers the result is, generally speaking, not an integer, and hence, this operation is omitted. Instead, the (04,n) order has been used for a rather different type of command, namely a question or test which asks whether the current contents of the A register are unequal to the current contents of integer storage register n? It is this question which is used primarily to decide whether we have been through a particular sequence of operations a sufficient number of times or whether we ought to keep on going through it again. This order is used only in connection with a "transfer" order.

11.4 Transfer Orders

The FLINT code is a sequenced language, that is, FLINT obeys the instructions in the order in which they are written down, unless a transfer instruction is encountered which tells the machine to go to some other location.

FLINT possesses two transfer orders -- (30,w) and (31,w). These orders behave identically as far as their transfer function is concerned, so they will be discussed together. (The 31 orders have additional properties which will be discussed later.) If, in the normal sequence of events, our FLINT code encounters an order (30,17) it will seek its next instruction at the beginning of word 17 of the current line of orders.

While the (30,w) instruction is sufficient for transferring to arbitrary words within one line, we must also have a method for transferring to arbitrary words within other lines. This clearly takes more information than we pack into one order, and so FLINT recognizes two kinds of transfers - short and long. For the long transfer it is necessary to specify to what line we wish FLINT to go next. As lines of FLINT coding are stored on the magnetic drum, a particular line of orders has an address consisting of (d) where d may be any integer from 0 through 11 and (ℓ) = any integer from 0 through 31. A long transfer, then, consists of two

successive instructions within the same word, having the general form (30,w) (d, c).

The need for long transfers and the necessity for transferring to the first instruction of a word imposes restrictions on the placement of transfer orders within our code. In particular, it requires that all transfer orders must be the last effective order in the word in which they are written. If, therefore, we write a transfer instruction in the first two positions of a word, the last two positions are wasted. Fortunately, instruction space is plentiful within FLINT and this need not worry us.

Both the (30,w) and (31,w) transfer orders are conditional on the state of Transfer Counter (TC). This counter is normally in a YES condition, but the (04,n) comparison order will cause it to be set to whatever the answer may be to the question it asks. Any transfer order will be obeyed if the TC is in a YES condition at the time the transfer order is encountered by FLINT. If the TC is NO, transfer orders will not be obeyed -- control passing to the first instruction of the next word -- but the TC is reset to YES.

11.5 Indirect Address Orders

In most digital computing we deal with two rather different types of data: simple isolated floating-point numbers such as constants, results of previous computations, and isolated coefficients -- and with large systematic groups of data, vectors or matrices, all of which are to be treated in some systematic fashion. The 31 floating-point storage registers, together with the arithmetic orders 10 through 19, suffice to treat the isolated individual numbers, but they clearly cannot handle large quantities of systematic data. To handle these numbers, we have the indirect address orders. They are very similar to the direct-address arithmetic orders:

(20,n) ((n)) —> X
(21,n) x + ((n)) —> X
⋮
(29,n) ((n)) / x —> X

except that ((n)), $0 \leq n \leq 30$, stands for the floating-point number whose address is presently stored in #n of the "integer" storage location. More exactly: when FLINT encounters the order 21.17, it will find in register 17 an integer somewhere between 0 and 1023, which it will then interpret as an address, and it will go to that address to find the floating-point number which it will then add into the accumulator. In the midst of this process it will also transfer the contents of integer register 17 to A.

These indirect-address orders are at once the most useful and the most difficult to use in the FLINT system, difficult especially for occasional users who have not previously been exposed to coding for our, or similar, machines. If compared with the corresponding procedure for "direct coding", the use of these orders is a simplification in as far as

(i) no distinction of left or right hand phases is necessary

(ii) if the same address appears in several orders and has to be modified in all of them, only one "integer" needs to be modified rather than all order words.

(iii) the reading of the address to be modified (into A) is automatic with the indirect address order.

Several attempts have been made to replace the indirect-address orders by another scheme; but none seemed simpler without sacrificing flexibility. This seems to be inherent in the interpretive system. In other words, only a compiler -- other than a word by word translater -- could bring real progress. Such a project, however, could not be achieved without much more time and manpower.

11.6 Non-Address Orders

So far, all orders have been divided into an operation and an address part, the latter one giving the address of one of the operands. The remaining orders 06.00 through 09.31 form an exception to this rule. The first group is used for multiplying or dividing, respectively, a floating-point number by an integer up to 31, which is directly given in the second pentad:

(06.k) x . k ⟶ X

(07.k) x / k ⟶ X

These orders have been useful for the coding of formulae containing small fixed integers, such as integration, differentiation and interpolation formulae, where they relieve the coder from the necessity of prestoring such numbers (as 2, 3, 4, 6 etc.) as floating-point numbers.

The second group will replace the number x, stored in the X accumulator, by a function of x. At present the list is short:

(08.00)	$2^x \longrightarrow X$
(08.01)	$\log_2 x \longrightarrow X$
(08.02)	$\exp x \longrightarrow X$
(08.03)	$\log_e x \longrightarrow X$
(08.04)	$\cosh x \longrightarrow X$
(08.05)	$\sinh x \longrightarrow X$

but an extension is easily possible by using the magnetic drum for the storage of such subroutines. A code for the four basic trigonometric functions has been written for, but not yet incorporated in, the FLINT system, and an algorithm for arctan x has been developed.

The third group is a somewhat accidental mixture of non-address orders which have proven to be quite useful for various purposes:

(09.00)	$1 \longrightarrow A$	(09.10)	$1 \longrightarrow X$
(09.01)	$a + 1 \longrightarrow A$	(09.11)	$x + 1 \longrightarrow X$
(09.02)	$a - 1 \longrightarrow A$	(09.12)	$x - 1 \longrightarrow X$
(09.03)	$\sigma \cdot a \longrightarrow A$ [1]	(09.13)	$\sigma \cdot x \longrightarrow X$ [1]
(09.04)	$-a \longrightarrow A$	(09.14)	$-x \longrightarrow X$
(09.05)	$a \leq 0\ ?$ [2]	(09.15)	$x \leq 0\ ?$ [2]
(09.06)	$a = 0\ ?$ [2]	(09.16)	$x = 0\ ?$ [2]
(09.07)	$a \geq 0\ ?$ [2]	(09.17)	$x \geq 0\ ?$ [2]

---

1) (09.23) will deposit the signum of x in a special register -- accessible only by (09.03) and (09.13) -- and leave the absolute value of x in X. (09.03) and (09.13) will multiply, without changing σ, the contents of A and X, respectively, with this signum.

2) These "questions" are used, just as the (04,n) order, to make the next transfer order -- which need not be the next order -- conditional.

(09.08) $2^a \cdot x \longrightarrow X$

(09.09) $a \longrightarrow X$; a integer

(09.20) $0 \longrightarrow X$

(09.21) $2x \longrightarrow X$

(09.22) $\frac{1}{2}x \longrightarrow X$

(09.23) $\mathrm{sgn}\, x \longrightarrow \sigma$ and $|x| \longrightarrow X$ 5)

(09.18) $\rho \longrightarrow A$ 3)

(09.19) $|x| \Longrightarrow a + x$; $0 \leq x < 1$ 4)

(09.24) $|x| \longrightarrow X$

(09.25) $x^{-1} \longrightarrow X$

(09.26) $x^2 \longrightarrow X$

(09.27) $x^{1/2} \longrightarrow X$

(09.28) reverses the position of the "transfer condition" from YES to NO, or NO to YES, respectively.

(09.29) is used only in connection with the 31 transfer orders. The only difference between a 30 and a corresponding 31 transfer is that the latter will cause FLINT to "remember" (Store) the location where the 31 transfer was given. The next (09.29) order will then transfer back to the beginning of the order word following the memorized address. This order, placed at the end of a subroutine, facilitates references to this subroutine at various places in the code (example: evaluation of $f(x,y)$ at the various stages of the RUNGE-KUTTA procedure) without the coder having to worry about changing the final transfer of the subroutine. This (09.29) transfer is conditional in the same sense as the other transfers, if preceded by questions, such as the 04,n or 09.05 etc. orders. As the subroutine referred to might itself refer to a "subsubroutine", care has been taken to provide for up to four levels: each 31 transfer will lead one step "down", each (09.29) one step "up".

---

3) (09.18) brings the exponent of x to A without changing the contents of X. Particularly useful for testing the order of magnitude of x.

4) This is about the most direct transfer possible from X to A since a must be integer. Used e.g. to find the "reduced argument" x for a periodic function.

5) see footnote (1) on previous page 11.6.0

(09.30) Read in decimal data from IBM cards. Each card holds up to two floating-point decimal numbers, together with their future addresses and in a format suitable for tabulation. (09.30) will cause conversion of such cards placed in the read hopper; the floating-point numbers will go first to X, then to the storage location specified in the "address" columns, while this address goes to A. This process will stop if an empty card or an empty half card is encountered. FLINT will then proceed to the next order.

(09.31) reconverts x into a floating-point decimal number which is stored in the form of a "card image" together with the decimal equivalent of a. Two such pairs (x, a) fit on one card and are automatically punched out as soon as a third (09.31) order is given. The format is the same as for input of numbers, as explained above.

Most of the functions carried out by the orders (06.00) through (09.29) could also be achieved by suitable combinations of the other orders; these non-address orders, however, add very much to the ease of coding for FLINT.

11.7 The "31" Address

FLINT makes provision for storing both floating-point and integer constants right in the order code, interspersed among the FLINT orders. In effect, the arithmetic orders invoking a 31 address tell FLINT to look in "the next possible space" -- and use it as if it came from a regular register. Thus (01,31) will cause FLINT to add into A the next "order" as a (binarily punched) integer. If we wish to multiply the contents of A by 3, we may write (03,31) (00,03), while (01,31) (02.03) will cause (A) to be increased by 2x32+3 = 67. The only restriction on the location of the integer is that it must be in the next order space after the instruction invoking it.

A direct floating arithmetic order with a 31 address obviously requires an entire word for its datum. Accordingly FLINT looks at the next entire word, interprets it as a floating binary datum, and uses it as instructed.

The FLINT order code then proceeds to the order following the 31-address order, with no skipping if this next order is in the same word. The word which has been used as a datum is skipped automatically, FLINT remembering that it is not a group of orders. (This automatic skipping holds true even if several 31-address orders are used consecutively, requiring several consecutive constants in the code. Thus, two floating arithmetic orders with 31-addresses do not refer to the same location, but to two successive locations.).

An indirect-address order with a 31-address will deal with the floating-point number stored at the address given in the next quarter word. This means, in effect, that the whole Williams memory, as far as it is not used by the interpretive code itself, is "directly" addressable by 20-bit orders. For example, (21,31) (13,17) will add the floating-point number stored in line 13, column 17, of the Williams memory to X and (25,31) (13,17) would store (X) at that position.

It seemed somewhat doubtful in the beginning whether the advantages which are afforded by the address "31" would be important enough to justify the logical complications involved;*) but after a year of use and experience this question can certainly be answered in the affirmative. As FLINT allows for the storage of only 30 integers (addresses) and thirty directly addressable floating-point numbers, storage place would soon be at a premium without the "31" addresses; while with this facility it has proved sufficient for much bigger problems than FLINT was originally designed for. Another aspect is equally important: the possibility of including constants right in the order code is a great help in the use of subroutines by cutting down their storage specifications to a minimum.

---

*) The additional code for all three applications, i.e. for fixed point, floating-point, and indirect address orders adds up to only 10 order words.

11.8 Illustrative Examples of FLINT Coding

(i) Computation of a Table for $J_3(x)$, $x = 0(0.1)1$.

Bessel functions for small arguments are best computed from their respective power series:

$$J_3(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!\,(p+k)!} \left(\frac{x}{2}\right)^{p+2k}$$

Five terms are sufficient for full accuracy if $x \leq 1$.

WORD	CODE	OPERATION	STORAGE
0	10.01	R ×	x in 00.01
	09.22	.1/2	$(\frac{x}{2})^2$ to 00.02
	09.26	x^2	
	15.02	$S(\frac{x}{2})^2$	
1	07.28	÷28	
	09.12	-1	
	13.02	$.(\frac{x}{2})^2$	
	07.10	÷18	
2	09.11	+1	
	13.02	$.(\frac{x}{2})^2$	
	07.10	÷10	
	08.12	-1	
3	13.02	$.(\frac{x}{2})^2$	
	07.04	÷4	
	09.11	+1	
	13.02	$.(\frac{x}{2})^2$	
4	13.01	.x	
	07.12	÷12	
	09.31	REC $J_3(x)$	
	10.01	R x	
5	09.31	REC x	
	09.18	ρ→ a	
	09.05	A ≥ 0 ?	
	06.10	.10	

WORD	CODE	OPERATION	STORAGE
6	09.11	+1	
	07.10	$\dot{\div}$10	
	15.01	S x	
	30.00	Tr	

(ii) Find a mod b, a and b integers

0	00.01	R a	a in 00.01
	00.00	—	b in 00.02
	00.00	—	a mod b to 00.03
	00.00	—	
1	05.03	S a mod b	
	02.02	-b	
	09.07	a $\geq$ 0 ?	
	30.01	Tr (to beginning of this word)	

(iii) Solution of the Quadratic Equation $x^2 + px + q = 0$

0	09.30	convert	q in 00.01
	09.22	1/2	p in 00.02
	09.14	.(-1)	
	15.03	S -p/2	
	09.23	sgn x => σ	
	09.26	x^2	
	12.01	-q	
	09.27	$x^{1/2}$	
	09.13	σx => x	
	11.03	+ (-p/2)	
	09.31	REC x_1	
	19.02	$\neg$ q	
	09.31	REC x_2	

(iv) Square Root of a Complex Number

$(p + iq)^{1/2} = a + ib$, p & q given , a & b = ?

Algorithm:

$$\left\{\left[\left(p^2 + q^2\right)^{1/2} + |p|\right] \cdot \frac{1}{2}\right\}^{1/2} \Rightarrow c$$

if $p \geq 0$: c => a, q/2c => b

if $p < 0$: c => b, q/2c => a

WORD	CODE	OPERATION	STORAGE
0	10.01	R p	p in 00.01
	09.23	sgn x => σ	q in 00.03
	09.26	x^2	a to 00.02
	16.03	Q q	b to 00.04
1	17.03	$+ q^2$	
	09.27	$x^{1/2}$	
	09.13	. sgn p	
	11.01	+ p	
2	09.24	|x|	
	09.22	. 1/2	
	09.27	$x^{1/2}$	
	15.02	S c as "a"	
3	15.04	S c as "b"	
	09.21	.2	
	19.03	$\sqrt{\ }$ q	
	09.00	1 => a	
4	09.03	. sgn p	
	01.31	(a + 3	
	00.03	=> a)	
	20.00	S q/2c as "a" or "b"	

11.9 Storage Space and Operation Times

Every interpretive subroutine is bound to reduce the storage space available for data and the speed of operation. This is the price paid for the ease in coding. We wish to show, in this last paragraph, how high the price of using FLINT is in these respects.

(i) Storage: Let us again divide the 1024 words of the Williams Memory into 32 lines (0 through 31) of 32 words each, which actually corresponds to the picture displayed on the screens of the memory tubes:

Line 31 is reserved for the "integers" as explained in 11.3; (31.00) = A;(31.31), which according to 11.7 cannot be directly addressed, holds the present order word.

Line 30 holds that line of pseudo code which has last been called (from the drum by a "long transfer") for execution. (30.31) holds the negative sum of (30.00) through (30.30) formed automatically by the FLINT read-in code. Checking of the sum of the whole line -- which must then be zero -- is automatic with each long transfer.

Lines 24-29 constitute the main part of FLINT, interpreting and executing all orders except (06.00) through (09.31).

Lines 21-23 are normally used for the execution of the orders (09.00) through (09.29); if, however, a conversion, (09.30) or (09.31), is ordered, the respective subroutine -- exactly 3 lines -- will be called from the drum to these 3 lines of the Williams Memory. The time lost by these drum to Williams transfers is negligible compared with the IBM card reading or punching times which are necessarily connected with these conversion orders.

Lines 18-21 are normally used by the functional subroutines (08.00) through (08.05); but if these are not needed in a code, the space may be used for data.

Lines 0-17 are free for floating-point data, but only line 0 can be addressed by the direct floating-point orders.

Lines 18-21 are normally used by the functional subroutines (08.00) through (08.05); but if these are not needed in a code, the space may be used for data.

Lines 0-17 are free for floating-point data, but only line 0 can be addressed by the direct floating-point orders.

In the present version of FLINT, the magnetic drum can only be used for the storage of code, but not for data without referring to "direct coding" for the transfer of such data to and from the drum. There is no good reason for this restriction and a corresponding change of FLINT is planned.

(ii) Operation Times: The interpretation and execution of a floating-point order in FLINT takes approximately 5 msec, while fixed point and non-address orders normally take a little less (3 to 4 msec). For comparison, machine multiplications and divisions last approximately one millisecond, addition and similar operations a little less than 0.1 msec. It thus seems that the use of FLINT would lengthen machine times by a factor between 5 and 50. Better estimates may be derived from the following considerations:

(a) The number of "mathematical" additions, i.e. excluding those for bookkeeping and address modification, is, in general, not too different from that of the multiplications and divisions together.

(b) The number of orders required to carry out a typical sequence of operations is normally much less in FLINT than in direct coding, even if fixed point coding is feasible, the average being about 1:2 (excluding FLINT subroutines such as 08.00, 09.27, 09.30, 09.31).

(c) The greater flexibility of the FLINT order code, especially the absence of scaling difficulties, often allows the use of more efficient mathematical methods, especially since coding space is practically unrestricted.

(d) The speed of in-and output is basically the same in FLINT and in direct coding; hence, if the time required for card reading and punching is an appreciable fraction of the total running time of the code, the time factor is further reduced.

(e) Debugging times for FLINT codes are, even with the inefficient present "TRACER", normally much shorter than in direct coding. Fewer errors are made and those made are more readily detected.

The result can be summarized as follows. The average effective slow-down factors are approximately:

Considering (a) alone	15
" (a) and (b)	10
" (a) through (c)	5-10
" (a) through (d)	2-10
" (a) through (e)	0.1-10

The last figure shows that the use of a floating-point interpretive routine need not always be wasteful of machine time. Experience has confirmed that it can even lead to a substantial saving.

12. ANOTHER FLOATING-POINT INTERPRETIVE ROUTINE (Code 1906)

Originator and Coder: Irving N. Rabinowitz

12.0 Introduction

1906 is a floating-point interpretive routine, similar in some respects to FLINT, but basically different in its logical structure. However, some parts of the two interpreters, notably the arithmetic operations are identical, and so will not be discussed in great detail.

1906 will accept a pseudo-code residing in the machine, and will interpret this, executing both arithmetic and logical operations. In addition, the structure of 1906 allows the coder to mingle direct code with his interpreted code with a minimum of effort. However, the orders available to 1906 are extensive enough so that this need not be done except to execute drum read or write orders, which are not provided for in the pseudo-code.

The pseudo-code for 1906 is kept in the Williams memory at all times, and thus can be modified by itself. In this respect the pseudo-code is closer to machine code than in the FLINT routine, the difference being that between a sequence-controlled machine and a stored-program machine.

12.1 Structure of the Pseudo-machine

As mentioned, 1906 is a stored program machine (and we might as well adopt the view that it is a pseudo-machine, since it avoids circumlocutions). The programmer has available to him without restriction 672 words (=21 lines) of high-speed storage, which can hold numbers, orders, or pseudo-orders. Numbers may be of two kinds, floating-point numbers, which represent quantities actually used in the computation, and fixed-point numbers, or integers, which are used for logical manipulations.

Since the arithmetic portion of the pseudo-machine is identical to that of FLINT, we need not describe the structure of floating-point numbers (see section 11.1).

Fixed point numbers, or integers, may be real numbers, i.e. signed forty bit machine numbers, which may be manipulated by a set of pseudo-orders, but which are better handled by direct coding; or they may be 10-bit numbers without sign for specifying certain logical parameters such as indices.

The structure of the pseudo-order words is similar to that of machine

order words in that two pseudo-orders occupy a single word, but the structure of an order is not (address, order), but rather (address, tag, order). The address is a ten-bit number representing addresses (00,00) through (20,31). The tag is a five-bit number whose individual bits specify certain B-registers. The order is a five-bit number. This allows a total of 32 orders, but by sacrificing the property of "B-modifiability" for some orders, namely those with order part 31, it is possible to increase the number of available orders to 63, which is sufficient for all purposes.

12.2 Arithmetic Orders

The arithmetic unit of 1906 consists of the pseudo-registers R and Q both of which hold floating-point numbers. The orders affecting these registers are the following: (Let (x) mean "the contents of x", where x may be a memory location, or R or Q.

A) B-modifiable:

24	Bring	(x) —> R
05	Add	(R) + (x) —> R
15	Subtract	(R) - (x) —> R
25	Subtract Absolute Value	(R) - |(x)| —> R
06	Multiply	(R) × (x) —> R
16	Multiply negative	-(R) × (x) —> R
07	Multiply & accumulate	(R)+(Q)(x)—> R
17	Multiply & accumulate negative	(R)-(Q)(x)—> R
08	Divide R	(R) / (x)—> R
18	Divide x	(x) / (R)—> R
26	Store	(R) —> x
27	Store zero	0 —> x; R unchanged
29	Load Q	(x)—> Q

B) Not B-modifiable, and address not used:

0031	Square root	$\sqrt{(R)}$ —> R
0131	Exponential	$e^{(R)}$ —> R
0331	Absolute value	|(R)| —> R
0431	Negative	-(R) —> R
0631	Square	$(R)^2$ —> R

C) Not B-modifiable, address used:

0531	Multiply (R) by address	(R) xN —> R
0731	Add (R) and address	(R) + N —>R

(In these two orders, N is one-fourth the address, so that the numbers from 1/4 to 127 3/4 are available for multiplication and addition without being stored. Negative numbers may also be used by complementing the address).

The nature of these arithmetic orders is self-explanatory, so we need not go into them further, except to note that they exist in the two classes of B-modifiable and not B-modifiable. These terms will be explained when we describe the logic of 1906.

12.3 Fixed-point Arithmetic

In addition to the pseudo-registers R and Q, there exists another pseudo-register called C, in which fixed-point numbers may be manipulated. The orders affecting C are not B-modifiable, but do use the address portion to represent a storage location. The orders are

0831	Bring	(x) —> C
0931	Bring negative	-(x) —> C
1031	Add	(C) + (x) —> C
1131	Subtract	(C) - (x) —> C
1231	Multiply	2^{+9}(C)(x) —> C
1331	Store	(C) —> x

These are all fixed-point operations except for 1231, Multiply, which, as in FLINT, includes a left shift of 9 places. Since manipulations on fixed-point numbers are primarily for logical, rather than arithmetic, purposes, the fixed-point operations will be applied to integers modulo 2^9, which in a machine word, of course, are represented as fractions. So that a product be properly scaled, the left shift of 9 places is included after multiplication.

12.4 Transfer or Jump Instructions

In order to change the sequence of operations, several types of transfers have been included in the order code. It is possible to change the

sequence of operations unconditionally by the order

10	Jump	Take next order from address and phase specified.

Note that jumps may be made to either order in a word. This specification is accomplished by a phase bit, which is part of the tag. In jump orders this bit is interpreted as specifying the feft-hand order of the location given by the address if the bit is zero, or the right-hand order if it is one.

The program may be stopped by

00	Stop and jump	Stop. If restarted, execute 10. However if the address is (00,00), go on to the next order.

Certain jumps may be conditional on the state of registers. Two such are conditional on the state of R, two on the state of C, and 16 on the state of 16"breakpoint switches".

The orders conditional on the state of R are

20	Positive conditional jump	If (R) $\geq$ 0, execute 10. If (R) < 0, no operation
30	Negative conditional jump	If (R) < 0, execute 10. Otherwise no operation.

It may be noted that there is no jump conditional on the state (R) = 0. The reason for this is that in floating-point arithmetic, round-off effects can cause troubles in this respect.

The two orders conditional on the state of C are:

1431 Jump if (C) is positive or zero.

1531 Jump if (C) = 0.

Note that here, since we are dealing with integers, a zero test is included, and is indeed a very powerful method of terminating loops, if the programmer is counting his way through a loop, a process which in general is unnecessary, due to the power of the B-registers.

The other sixteen conditional jump orders are variants of the order

23	Breakpoint stop and jump.	If "breakpoint switch" n is 1, execute 00; if it is 0, no operation.

Ideally, of course, the breakpoint switch would be a true mechanical switch which the operator could turn to an "on" or "off" position. However, what

is actually done is to reserve a word in storage for this purpose. The first sixteen bits of this word are the 16 breakpoint switches, and are manually made 1 or 0 by the operator, at the instruction of the programmer. When the interpreter encounters the order (a,ϕ,n,23), it inspects bit n of the breakpoint word. If it is a one, the order 00 is executed, i.e. it stops, and upon restarting executes a jump to the order at phase ϕ in word a. If it is a zero, the interpreter ignores this order and proceeds to the next order. The purpose of this order is to aid in debugging a program. The programmer can insert 23 orders at various points of his program, which will transfer to a print-out routine which he has written. When first running his program, he can set all the breakpoint bits to one, so that the program will stop at each 23 order. After the print-out, the first bit in the breakpoint word is made zero, and the program continued. Now the machine will not stop at the order 0023, but will stop at the next breakpoint stop and jump encountered. The process can then be repeated with all the breakpoints in the program, thus giving the programmer information as to the state of the memory at intermediate points of the computation. When the program is finally debugged, it need not be changed at all, since the 23 orders have no effect on the execution of the correct program, the breakpoint switches not being set.

12.5 Other Logical Instructions

Two other logical instructions are available to the programmer:

21 Set return	If this order is encountered in word a, then in the phase and address specified, there is written the half-word < a+1, 10 >
22 Exit	Exit from interpretive mode to direct coding.

Order 21 is designed for use with closed subroutines. Symbolically, its use is as follows: when a closed subroutine is to be used from word a, the programmer writes <Exit, 21/Entry, 10 >, the effect being that when the closed subroutine has been executed, and control has arrived at the exit word of the subroutine, a jump is made back to the main program.

The order 22 is used to go out of the interpretive mode into direct coding, and has the form of a 10 order, except that the order at the specified

address and phase is not interpreted, but is executed by the real machine. To re-enter the interpreter, the usual type of link-planting order pair is used, i.e., to re-enter from word a, the programmer writes < a CP/3100 U1 >, in direct machine code.

12.6 The B-registers and their Instructions

We now come to the most powerful part of code 1906, the B-registers. These are a set of registers labelled B1, B2, B4, and B8 (or B1, B2, B3, B4, if desired. The 1, 2, 4, 8 notation is a more mnemonic one, as will be seen), which have the capabilities of counting and modifying addresses, so that loop formation is reduced to the writing of two instructions. It will be noted that the arithmetic orders (§12.2) were divided into several groups, one of which is that of the B-modifiable instructions. For these instructions, the tag part of the order plays a role. The tag consists of five bits, four of which refer to B-registers. It has the form 0 b_8 b_4 b_2 b_1, where b_i refers to Bi. The reference is the following: if b_i is one, the address used by the pseudo-machine is not the address written in the order, but is this address plus the contents of Bi. This is the effective address. The use of this device is to allow references to arrays of numbers without having to go through the process of counting and address modification within a loop. If more than one b_i is one, the effective address is the original (written) address plus the contents of all the Bi referred to by the b_i. In this way it is extremely convenient to refer to two- or three-dimensional arrays.

Besides their address-modification properties, the B-registers have counting properties which are used to go around loops the proper number of times, and to exit from the loop when it has been executed. This is accomplished by the use of two types of orders, the SET and TEST orders. These are

01 Set B1	Each of these is a three-address order, affecting the B-register mentioned. These three addresses
02 Set B2	are those of I, Δ , and F, respectively, and have the effect of setting I, Δ, and F into three
03 Set B3	storage locations, the totality of which constitute the pseudo-register Bi.
04 Set B4	I is the initial value of an index, Δ is its increment, and F its final value.

11	Test B1	These orders test the B-register referred to in order to see whether the loop has been traversed
12	Test B2	$\lvert(F - I)/\Delta\rvert$ times. If this is the case, then control passes to the next order in sequence. If not
13	Test B3	then the contents of the B-register are increased by Δ (which may be positive or negative), and
14	Test B4	a jump is made to the address and phase specified, in order to traverse the loop again.

The use of these orders is as follows: as an example, let us consider the simple problem of adding up the numbers located at storage locations a, a+1, a+2, ..., a+99. The simple-minded method is, of course, to write down 100 orders of the form

```
  a   Bring
a+1   Add
a+2   Add
  .
  .
  .
a+98  Add
a+99  Add.
```

There are obvious disadvantages to such a course. The more usual procedure is to set up a loop to accomplish the same thing. In machines without address-modifying capabilities, this loop is accomplished by setting a counter to zero, using it to change the address of an order, executing this order, testing the counter against its final value, and either increasing it and going back to the beginning, or leaving the loop. In the interpretive mode, this is accomplished more simply by the use of the B-registers. For the problem mentioned, we could proceed as follows: assuming that R, the pseudo-accumulator is clear, we have

```
    x   SET B1   <0>, <1>, <99>        (Notation: The symbol <x> means
                                        "the location of x.")
(x+1)L  ADD      <a_0>, b_1
(x+1)R  TEST B1  x+1,L.
```

This two-word loop does the following: in word x, the SET order takes the contents of the three locations specified, namely 0, 1, and 99, as integers scaled 2^{-9}, and deposits them in the three storage locations which make up B1, the location to which I = 0 goes being used internally to 1906 as the counter for the loop. Thus the first value for the index is taken as zero. The left half

of word x+1 contains the order to add the next value of a_i into the accumulator. The address written is that of the first value of a_i, namely a_o, but the effective address is a_o + (B1), since the b_1 bit is in the tag part of the instruction. The first time through the loop, (B1) = i = 0, so that a_o is brought to R. At word x+1, right phase, is the order to terminate the loop, TEST B1. The sequence of events that occurs here is as follows: the loop counter is compared with 99. If they are unequal, it is increased by 1(the preset value of Δ), and a jump is made to the order at location x+1, left phase, namely, the ADD order. However, if they <u>are</u> equal, the incrementing and jumping do not take place, and the next order to be executed is the order at location x+2, left phase.

Thus the effect of the loop is to add a_o, a_1, . . ., a_{99} into the pseudo-accumulator, and then exit from the loop, with the sum in R.

The structure of such loops has an almost exact counterpart in the mathematical notation used for these operations. For the problem mentioned, one might write

$$b = \sum_{i=0}^{99} a_i ,$$

to describe the formation of the sum. Another way would be to say

$$(R) + a_i \longrightarrow R \quad \text{for} \quad i = 0(1)99,$$

which has its exact counterpart in the code. The code, in symbolic terms, could have been written

```
    x       SET i FOR    0(1)99
(x+1)L      ADD  a_i
(x+1)R      TEST x+1,L      .
```

More than one B-register may be used in a single loop. Suppose that the problem involved is to add two vectors a_i and b_i. Suppose that the vector a_i is arranged in the memory in locations a_o, a_o+1, a_o+2, . . . , a_o+n, i.e. in consecutive locations specified by $\langle a_i \rangle = \langle a_o \rangle + i$, and that the vector b_i is stored differently, e.g., in every other location, starting at $\langle b_o \rangle$

and such that $(b_i) = <b_o> + 2i$. It is necessary to use <u>two</u> B-registers for address modification in the same loop. Let us write the mathematical discription of the problem:

$$c_i = (a_i + b_i), \quad \text{for } i = 0(1)n$$

Assuming that the c_i are stored sequentially starting at $<c_o>$, we rewrite this in the following form:

$$c_i = a_i + b_i, \quad \text{for } i = 0(1)\ n \text{ and } i' = 0(2)2n.$$

The code is then a transcription of this statement to a vertical format:

```
      x   SET i = 0(1)n
    x+1   SET i' = 0(2)2n
  x+2,L   BRING  a_i
  x+2,R   ADD    b_i
  x+3,L   STORE  c_i
  x+3,R   TEST i, x+4,L
  x+4,L   TEST i', x,2,L
```

Note that although there is only a single loop, yet there are two SET orders and two TEST orders, since two B-registers are involved. Since the indices are to be counted synchronously, two TESTS must be made for each traversal of the loop. The first TEST increases the index i, while the second TEST is used to escape from the loop when finished. Thus the address of the first is x+4,L, so that regardless of whether or not the loop is done, the second TEST is executed whenever the first is.

In dealing with two-dimentional arrays such as matrices, it is desirable to be able to modify an address by <u>two</u> indices. As mentioned above, this is possible by including more than a single 1 in the tag. Consider the problem of the multiplication of two square matrices. Mathematically we write

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik}b_{kj} \qquad \text{for } i = 0(1)n\text{-}1 \qquad j = 0(1)n\text{-}1$$

However, we must take into account the fact that the indices are not all increased by unity, since the matrices may be stored in the memory in such a way as to require different modification. If we assume that the matrices are

stored in the memory so that the elements of the matrix, read from left to right and top to bottom, occupy consecutive locations, e.g., a_{oo}, a_{01}, . . . , a_{on-1}, a_{lo}, . . ., $a_{1,n-1}$, . . . , $a_{n-1,o}$, $a_{n-1,1}$ ..., $a_{n-1,n-1}$, then a simple description of the location of any element is

$$\langle a_{ij} \rangle = \langle a_{oo} \rangle + i + nj$$

Thus, for the three matrices involved, we have

$$\langle a_{ik} \rangle = \langle a_{oo} \rangle + i + nk$$
$$= \langle a_{oo} \rangle + i + k'$$
$$\langle b_{kj} \rangle = \langle b_{oo} \rangle + k + j'$$
$$\langle c_{ij} \rangle = \langle c_{oo} \rangle + i + j' \quad ,$$

and thus there are four indices which must be used, namely, i, j', k, and k', such that $i = 0(1)n-1, j' = 0(n)n^2, k = 0(1)n-1$, and $k' = 0(n)n^2$. We may then write the code as

```
  x     SET   i = 0(1)n-1
 x+1    SET   j' = 0(n)n^2
 x+2,L  BRING zero          (This order has the effect of clearing
 x+2,R  BRING zero           R prior to accumulating products. The
                             second BRING is just a skip in order
                             to get into the proper phase.)
 x+3    SET   k = 0(1)n-1
 x+4    SET   k' = 0(n)n^2
 x+5,L  LOAD L  a_ik'
 x+5,R  MULT&ACC  b_kj'
 x+6,L  TEST  k, x+6,R
 x+6,R  TEST  k', x+5,L
 x+7,L  STORE   c_ij'
 x+7,R  TEST  j',  x+2,R
 x+8,L  TEST  i,  x+1,L  .
```

This code does the following: the orders at x and x+1 set up the B-registers for i and j' for the i and j' loops, and the orders at x+7,R and x+8,L close these loops. The effect is that the inner loop (j') runs across the row for which the index i has a certain value. When the end of the row is reached, as signalized by a failure of the TEST at x+7,R, the i loop is tested. If the loop is not done, the row index, i, is increased and the next row of c_{ij} is computed. Within these two outer loops is a single inner loop, which however, involves the use of two B-registers, one to count k and the other to count k', in a fashion similar to the problem of vector addition given above. Prior to entering the loop, R is set to zero by the orders at x+2. The reason that two such orders are given is that the SET order occupies a full word so that we must skip an order after the first BRING to get to the left half of the next word. The inner loop consists of the setting for both k and k', the actual computation of the (i,j) element of the result matrix, the closing of the k and k' loop, and the storing of c_{ij}. The actual computation of c_{ij} takes place at x+5, where the partial sum $\sum_{k=0}^{k} a_{ik}b_{kj}$ is accumulated, i.e., the vector product of row i of a by column j of b. Note that here again, the TEST at x+6,L is used only for stepping the k index up, while the actual testing for end-of-loop is done at x+6,R.

12.7 Integer Manipulations and Non-Automatic Modification

It may be noted that in the examples in §12.6 we were dealing only with indices, while addresses were given in the word referring to the operand. It is entirely possible to use the B-registers for carrying actual addresses, so that the code itself need have no addresses of operands in it at all. This is sometimes a convenience, for example, in changing codes to accomodate larger arrays than was originally contemplated. Thus, if a programmer has been using 5x5 matrices stored sequentially, and has to go to 6x6 matrices, the code itself would have to be changed if the addresses written in it were the locations of the first elements of the 5x5 arrays. This difficulty may be avoided by the following subterfuge: rather than dealing with indices and considering expressions of the form i= I(△) F we can deal with addresses in expressions of the form i = A+1(△)A+F, where A is an address which is stored in some

location. The loop setup would then involve forming the integer sums A+I and A+F, where A, I, and F are preset integers, and SETting the B-registers with these numbers, rather than with the values of the indices alone. The reference to the operand a_i would then be (say) ADD 0, b_i rather than ADD $\langle a_o \rangle, b_i$, and it would become unnecessary to change the code itself to make changes in the placing of operands.

Occasionally it may be advantageous to modify addresses in a coded fashion, rather than using the automatic features of the B-registers, for example, if one were dealing with expressions which might involve the use of more than four B-registers at once. Such might be the case in computing an expression of the form

$$c_{ij}^{(m)} = \sum_{k=0}^{n} a_{ik}^{(m)} b_{kj}^{(m)}$$

for i, j = 0(1)n, and m = 0(1)M. All four B-registers would be tied up during the inner loops of the computation, so that it would be necessary to have a separate counter for m, which could be added into the settings of the B-registers during the inner loops, i.e. one could compute the numbers A+I+(M+1)m and A+F+(M+1)M before setting the B-registers in the inner loops. Alternatively, the actual orders could be modified by adding (M+1)m to their addresses prior to setting the B-registers. This can also be accomplished by means of the fixed-point instructions, and the jumps conditional on the state of C would be used to determine when the m-loop has been completed.

12.8 Decimal Input and Output

There are two orders which affect the input and output of floating-point numbers,

09	Read	Read n cards, each with two words, into locations specified by address, sequentially. n given by the tag, $1 \leq n \leq 32$.
19	Punch	Same, in the other direction.

The card format consists of two words per card, each half of the card containing a single floating-point number. The format of a number is almost

completely free, with the exception that the sign of the number must occupy the first column of the field, and all numbers must have a decimal point. Thus, for example, to input the number 3.14159, it is not necessary to punch it as 3141590000 01, but it may be punched directly as 3.14159, providing only that the first column of the field is blank, for the sign. To input both mantissa and exponent, e.g., the number 3×10^{10}, it may be punched as ∅3.X10, where ∅ indicates a blank column. This notation indicates the mantissa as 3, terminated by a decimal point, and the X10 represents the exponent.

On output the number appears in a standard fashion, namely in the form ± 3.141590000X00. Output numbers may, of course, be later re-used as input, since the output format is acceptable to the input routine.

12.9 Tracing Mode

In order to help debug interpretive programs, a tracing mode has been added to 1906, and is operated by using a special card preceding the deck. This card contains two addresses punched by the user, the address at which tracing is to start, and the address at which it is to end. The card also includes certain standard coding which puts 1906 into the tracing mode (by changing a transfer instruction within 1906). The pseudo-code is executed at full speed (except for a few orders involved in the testing of addresses) until the starting address for tracing is encountered, at which point each instruction executed is accompanied by a card specifying the contents of R and of the B-register involved, if any, and also, of course, the location of the order itself. This procedure is carried on until the location counter of the interpreter becomes larger than the final address specified, at which point tracing is stopped and the code continues to be executed at full speed.

In this way, it is possible to debug sections of a program by tracing the earlier portions of a program first, verifying that they are correct, and then tracing later portions without being forced to repeat the trace of the earlier parts, which saves a great deal of machine time.

13.0 SERVICE ROUTINES FOR DECIMAL IN- AND OUTPUT

In order to help save time in the coding and debugging of problems, several routines of general utility were written. These routines were designed to be used in conjunction with the programmer's own code to help out in the processes of input and output of decimal numbers, and in the detection of errors in a code. The routines described here were written at different times, and are therefore quite independent of each other. It is planned to rewrite them in an integrated form so that any service routine can be called in a standard manner with a minimum of special handling.

13.1 GENERAL DECIMAL INPUT ROUTINE

Originator and Coder: Irving N. Rabinowitz

This is a short routine which allows a programmer to dispense with most of the "bookkeeping" involved in decimal-binary conversion. It is written in the form of a closed subroutine which uses three pre-set parameters for the determination of the arrangement of digits in the cards. Each time the subroutine is entered, it has the effect of reading the next number from the card, translating it into binary, and delivering it to the main program in the accumulator. The three parameters necessary for the routine are: D, a "stencil" or "mask" word which has one's in those stages where digits are punched on the card, and zero's elsewhere; S, a similar word which specifies the columns where signs appear on the card; and C, the number of cards of input. As an example, if the card format consisted of five seven-digit numbers and signs, filling all forty available card columns, we would have

D = 0111111101111111011111110111111101111111,

and

S = 1000000010000000100000001000000010000000.

The fields of each card need not be the same size, but all cards must have the same format. While the routine does not count, it does signal the end of the input by the state of the Q-register. If this is non-zero, then the input is finished, otherwise there is more to come.

Since it is written in the form of a closed subroutine, the programmer has complete freedom as to the placing of the input in the memory, the scaling of the input, and the sequence of reading from cards. Thus, for example, the

programmer could read in a vector, operate on it, store it on the drum, and then start over with the next vector to be read from cards, without concerning himself at any point with the decimal nature of the numbers on the cards.

13.2 GENERAL DECIMAL OUTPUT ROUTINE

Originator and Coder: Irving H. Rabinowitz

This routine was designed to allow almost complete freedom of output format without the necessity of the programmer's counting or bookkeeping. The output format is described as a linear function of two variables, called i and j, and various numbers specifying number length, presence or absence of signs, etc. The routine does its work essentially by setting up a correspondence between the given arrangement of binary data within the machine and the desired arrangement of data on cards. To output a group of data, it is necessary to specify six words of information. These are

1) The location and arrangement of the binary data on the drum. This word consists of the coefficients of the formula

$$L(n_{ij}) = L_{oo} + ai + bj$$

The routine can then use this word to find the binary data prior to conversion.

2) The card number (counting from zero) to which the decimal version of n_{ij} is to go. This word consists of the coefficients of

$$C(n_{ij}) = C_{oo} + ci + dj$$

3) The column of the card in which the decimal number (including its signs, if any, and its decimal point, if any) is to start. Again, this word consists of the coefficients of

$$K(n_{ij}) = K_{oo} + ei + fj$$

4) The limits of i and j. These are the numbers I_o, I_1, J_o, and J_1 for which $I_o \leq i \leq I_1$ and $J_o \leq j \leq J_1$.

5) A word specifying the representations of the plus sign, the minus sign, the decimal point; also the number of digits of the output, and the number of digits in front of the decimal point.

6) Finally, a word telling how many cards are to be punched out.

As an example to illustrate the use of the routine, let us consider the problem of converting a 5x5 matrix from binary to decimal. Suppose that the matrix is stored on the drum at locations (08,04,00) + j + 5i, i.e. the elements of a row are in consecutive locations, and the various rows follow each other consecutively. Since we would like to tabulate the output in the form of a matrix, let us put the five numbers of a row onto a single card, row 0 going to the first card, row 1 to the second, etc. Thus we see that the card number is simply $C(n_{ij}) = i$. Since we have five numbers to put on a card, we can break the card up into five fields, each eight columns wide, starting at columns 0, 8, 16, 24, and 32. These fields will then represent a column j of the matrix, and we can say that the starting card column number for matrix column j is $K(n_{ij}) = 8j$. Since we have a 5x5 matrix, we obviously have $I_0 = J_0 = 0$ and $I_1 = J_1 = 4$. Now we must decide on the format of the individual numbers. Let us suppose that we will want a blank column to represent a plus sign, a "12" punch to represent a minus sign; that we do not want a decimal point, and that we want four decimal places in the answer. Then the fifth word will contain the information that R(+) = $\emptyset$, R(-) = "12", R(.) = $\emptyset$, D =4, where R(x) stands for "the representation of x". Finally, we will want to punch out five cards.

The six words will then be punched onto cards (in binary) as follows:

1)	00/08,04,00/00,05/00,01	Location
2)	00,00/00,01/00,00/00,00	Card number
3)	00,00/00,00/00,08/00,00	Column number
4)	00,00/00,04/00,00/00,04	Limits
5)	00,00/00,12/00,00/00,04	Representations and length
6)	00,05/00,00,00,00,00,00	Number of cards to be punched

The programmer then inserts this card into the code deck and loads the code, assuming that the binary data is already stored on the drum, and out comes the decimal data.

This is a particularly simple example of the use of the routine. It has provisions in it for allowing the programmer to write a "de-scaling" routine, to call the output routine from the drum, to have the output routine handle as many as 22 different groups of output cards, and to have complete

control of the process of output, even to the point where he may, in a coded manner, change the parameters of the output.

The code has a fairly complete search for consistency among the parameters so that, for example, the programmer cannot ask for more than forty columns of information on a single card. Stops are provided at these error discoveries with sufficient information to allow the programmer to either find his error easily off the machine, or, if he recognizes it immediately, to correct it by simply putting the correct parameter word into the machine via the accumulator.

It is hoped that a new version of the General Decimal Input and General Decimal Output will be written in the near future, and that it will then no longer be necessary for a programmer to concern himself with these processes.

14.0 ASBY - AN ASSEMBLY CODE

Originator and Coder: Bryant Tuckerman

The necessity for some sort of assembly code to be used in conjunction with an automatic computer has long been recognized, for such a code has the properties that not only is coding and debugging made simpler, but it allows the building of a library of subroutines which may be incorporated into any code, without the necessity for readdressing. Furthermore, if it is necessary to change a code, the fact that it is written in a language more general than real machine code is a great advantage, since changes to one region of a code need not in general affect other regions.

The ideal type of assembly code is of course that one which takes the mathematics and English that the coder writes as his statement of the problem, and turns this into machine coding without the necessity for human intervention. However, such codes are extremely difficult to accomplish, and, indeed, their theory is not fully understood. It is therefore necessary to restrict ourselves to somewhat less sophisticated types of assembly routines. Among other considerations that must be applied are those of time and space. Since machine time is a valuable commodity, the assembling of a code written in some external language should be only a very small proportion of the running time of the assembled code. Furthermore, since space is at a premium, the secondary storage of the machine being finite (and for some applications, quite small), it is desirable that the input, or unassembled version of the code not be much larger than its final version. In order to reach these objectives, certain compromises between efficiency of machine use and ease of coding must be made.

14.1 Format of Subroutines

ASBY assumes that all coding consists of subroutines written in a special language called Format C which is almost machine language. In general, addresses are relative, i.e. they consist of a tag identifying the subroutine and a number which indicates the address within that subroutine. Addresses referring to words within the present subroutine are all larger than (16,00),

i.e. all subroutines are written as if they were to be operated from a block of consecutive memory locations starting from (16,00). Furthermore no subroutine may be longer than 8 lines (256 words). Addresses referring to other subroutines are smaller than (16,00) and are subdivided into 14 categories, according to the first pentad of the address with 0,1 and 2 being treated as a single category. The addresses beginning with 0,1,2 are treated as absolute addresses and are never modified by ASBY. The other thirteen categories of addresses begin with the integers 3 through 15, which may be considered 13 tags identifying 13 other subroutines. The rest of the address is a pentad (0 through 31), thereby limiting all references to other subroutines to the first 32 instructions in these subroutines. This is no real restriction for logical operation since it is difficult to imagine a subroutine which will require more than 32 different types of external references to it; usually 2 or 3 will suffice. Data, on the other hand, must also be organized into our subroutine format if they are to be assembled together with the code and here the limitation of only 32 distinct references may require some indirect addressing system for blocks of data. Data subroutines can, of course, be 256 words long.

While any one subroutine will probably not need to refer to more than 13 other subroutines, nevertheless we shall very quickly acquire a library of considerably more than 13 subroutines with many mutual cross references. Thus it is not practical to limit a tag (say tag 5) to refer always to the same subroutine (say SINE of X) whenever it is used. Format C therefore provides a tag-defining group of words which sits at the front end of the subroutine so long as the subroutine remains in the library, but which is removed from the subroutine after serving its purpose during assembly. This tag-defining group consists of:

1) a single parameter-word, PARW, having ones in those binary stages corresponding to the tags used in this subroutine, and zeros elsewhere. (If we use tags 3,4 and 7 then PARW is 000110010...0.)

2) a sequence of IDENTifing words, one for each tag used, arranged in increasing numerical order of the tags. (In our example above, PARW would be followed immediately by three IDEN words, the

first bearing the identification of that subroutine to which we are referring by the tag 3, the next identifying the subroutine we are invoking with tag 4, and the third identifying the subroutine we refer to by the tag 7.)

In addition to these tag-defining words, a one-word description, DSCR, of the current subroutine also appears in the front-end of the library form of all Format C subroutines. This word specifies the Williams assembled length, dw, of the subroutine; the library (Format C) length, $d\ell$, of the subroutine; and the number of constants, c, at the end of the subroutine which are to be left unmodified. Each of these discriptive numbers occupies 10 bits. We may summarize this form by

$$DSCR = (dw;\ d\ell\ ;\ c;\ 0)$$

All this front-end information of ASBY is placed between words (16,00) and (16,01) and none of it acquires any location address since it will ultimately be removed. Word (16,00) is the full word IDENtification of the subroutine and the actual code begins in (16,01). Thus we have our complete Format C outline:

(16,00)	IDEN	( )
	DSCR	(dw; $d\ell$; c; 0)
	PARW	( )
	$IDEN_a$	( )
(16,01)	code	( )
( )	end code	( )
	-SUM	( )

The word designated by-SUM is for checking purposes, and contains the negative of the sum (modulo 2) of all the other words of the subroutine. No gaps in the code within a subroutine are permitted and all subroutines are punched in straight binary, 12 words per card except possibly at each end where an entire card may not be needed. Routines may, of course, be punched as decimal pentads and then converted and punched out in binary by using

existing codes. The total number of non-William memory words is 3 + t where t is the number of tags used, hence

$$d\ell = dw + 3 + t$$

A special feature of ASBY allows us to define fields of variables of arbitrary but given length, dw, by subroutines occupying only $d\ell = 4$ library words, in the form

```
(16,00)  IDEN  (              )
         DSCR  ( dw; 4; 0;0   )
         PARW  (      0       )
         -SUM  (              )
```

This is assembled as dw words, the first being IDEN, the rest 0 or trash according as the drum-image of Williams was initially cleared or not.

(More generally, any number of irrelevant or zero words in the end of a subroutine may be omitted from the library form to become 0 or trash upon assembly, by proper choice of $d\ell < dw + 3 + t$.)

14.2 Directory Format for ASBY

In addition to his library of subroutines the customer must also supply a Directory which must include at least the IDEN of the first routine to be obeyed. In general, no further information need be supplied,*) since ASBY will look at this routine, will see what others are called for, will examine them in turn, etc. until it has constructed the complete directory during assembly.

---

*) A non-zero starting Williams address should, however, be supplied by the customer for the location of the first subroutine as w of its INFO word. Otherwise the first routine will go into (00,00), thereby violating the convention that we will keep the bottom 12 words of the Williams memory free for small service routines.

The customer may, however, override part of this process by supplying the IDEN and assembled location of any of the other subroutines he wishes to mention. (This facility may turn out to be useful during debugging, since he will then know that certain crucial subroutines begin in some easily remembered locations. It may also be used to cause the loading of auxiliary subroutines not directly needed by the code but which would be valuable to have available in the Williams memory during debugging.)

The final form of the Directory consists of pairs of words

$$\begin{array}{ll} \text{IDEN}_1 & (\qquad\qquad\qquad\qquad) \\ \text{INFO}_1 & (\ w;\ d\ell\ ;\ u;\ \ell\qquad) \\ \text{IDEN}_2 & (\qquad\qquad\qquad\qquad) \\ \text{INFO}_2 & (\ w;\ d\ell\ ;\ u;\ \ell\qquad) \end{array}$$

where w (10 bits) = Williams address of the assembled subroutine
$d\ell$ (10 bits) = library length of the subroutine
u (5 bits) = unimportant and irrelevant
ℓ (15 bits) = drum address of the subroutine

If the customer wishes to fix the location of a subroutine, he need only list its IDEN and then supply its w (the other information will be constructed correctly by ASBY even if supplied here incorrectly.)

In the absence of w, ASBY will supply it as the next consecutuve location. However, at the conclusion of assembly the completed Directory is punched out and control passes to the RELOAD code.

II. SOLUTION OF PROBLEMS

20. General Remarks

During the period covered by this report the emphasis has not primarily been placed on "production" but rather on exploring the potentialities of the computer as a research tool in a variety of problems and applications, while our engineering group was engaged in improving, hence changing our equipment. As a result, no systematic effort was made at the time to build up a library of subroutines, or to standardize coding procedures, and the number of problems solved is comparatively small.

It would not be easy to arrange the problems treated on our machine into clear-cut groups; three such groups can be distinguished, however, and we shall deal with them separately.

The first should be headed METEOROLOGY. Dr. Charney's group used about one third of the total machine time available for operation during the period covered by this report. Its problems and results will be reported under the terms of Contract Nonr 1358-(02) (between the Institute for Advanced Study and the Office of Naval Research), by which the meteorological research and machine coding were supported.

The second goup of problems deals with ASTROPHYSICS. All of this work has been directed by Prof. Martin Schwarzschild (Princeton University), either directly or via his co-workers. These problems are discussed in chapter 21, to which Prof. Schwarzschild was kind enough to write an introduction.

The problems in the third group are only loosely related to each other by virtue of their common subject matter, viz. ATOMIC AND NUCLEAR

PHYSICS. No two problems have originated with the same individual. It seemed advisable, however, to unite them in one chapter (22) so that the reader interested in physics can find them more easily.

The last chapter (23) is nothing but a collection of those problems which do not fit into any of the other groups. Size and importance of these problems are as varied as their subject matter. The Historical Ephemeris has taken close to one man-year of coding time (nearing completion at the close of this report) and will be of lasting value. The remaining problems are considerably shorter, and a small number of minor codes, without any special features, have been omitted. They have been helpful, however, in demonstrating the importance of library subroutines and determining their specifications.

21. ASTROPHYSICS

Astrophysics differs from other physical sciences by the circumstance that its objects of study can be observed but cannot be experimented with. This impossibility of physical experimentation can be compensated for to a remarkable degree by numerical experimentation. It is for this reason that numerical research plays such an unusually important role in astrophysics.

The problems here investigated are of two types; the first type refers to the undisturbed internal structure and evolution of stars,[1] whereas the second type refers to perturbations of stars such as stellar pulsations[2] or hydromagnetic waves.[3] The first type leads to highly non-linear, high order eigenvalue problems. In them the main difficulty consists in finding efficient methods of determining the eigenvalues by trial and error. The second type leads to partial differential equations with as much as three independent dimensions (two in space and one in time). Here the main difficulty consists of finding efficient methods for the numerical intergration that are free from numerical instabilities.

---

1) cf. §§ (21.10), (21.40)

2) cf. §§ (21.20), (21.50)

3) cf. § (21.30)

21.10 NUMERICAL EXPERIMENTATION ON STELLAR EVOLUTION

Originator and Analyst: M. Schwarzschild
Coder: Mrs. H. Selberg

Recent progress in nuclear physics has made it possible to determine fairly accurately the reaction rates, as functions of temperature and density, of those nuclear processes which provide the main energy sources in the stellar interior. These reaction rates have provided the last link necessary to formulate the problem of stellar structure and evolution in a unique manner. Consequently, it is now possible, in principle, to derive all physical characteristics, such as, for example, central temperature and total luminosity, for a star of given mass and initial composition as a function of time throughout the star's evolution.

In practice, however, the solution of this problem is complicated, both as regards the methods of numerical analysis to be applied, and as regards the effective carrying through of the large amount of numerical calculations. The difficulties arise largely from the fact that in the space co-ordinate (from the center to the surface of the star) the physical conditions present a highly non-linear system of differential equations of the fourth order with two boundary conditions at the center and two at the surface, but also from the fact that the energy release by gravitational contraction gives the problem in the time co-ordinate the character of a heat conduction problem with its well-known danger of numerical instability.

The numerical solution of problems in the theory of stellar evolution is clearly of importance to theoretical astrophysics. Simultaneously, the development of methods for the solution of this problem appears likely to be valuable for the solution of a large class of non-linear high order boundary value problems.

21.11 Qualitative Description of Stellar Evolution

Consider a star of a given mass and a given initial chemical composition. Assume that the matter of which the star consists was originally well mixed so that the star starts out chemically homogeneous. After a pre-stellar contraction

phase a star will settle into its inital equilibrium configuration in which all the hydrostatic and thermal equilibrium conditions are fulfilled. In particular the internal temperatures will be just right so that the hydrogen burning produces energy at a rate exactly compensating the losses by surface radiation.

The star would remain indefinitely in this particular equilibrium configuration if it were not for the circumstance that in time the nuclear burning alters the chemical composition of the deep interior, which in turn causes changes in the equilibrium configuration. The investigation of stellar evolution aims to derive the sequence of configurations through which a star evolves and, in particular, to compute the changes of the luminosity and the radius of a star during its evolution. A comparison of these two quantities with observation provides a check on theoretical developments.

A previous investigation*) has suggested the following sequence of evolutionary events for a star with a mass similar to that of the sun. In the initial phases more and more hydrogen in the core is transmuted into helium. After a while the hydrogen in the core is exhausted and the hydrogen burning continues in a shell further out. The helium core becomes isothermal. It steadily grows in mass but shrinks in size while simultaneously the envelope of the star becomes more and more expanded. The contraction of the core produces steadily increasing degeneracy of the gases near the center while the growth of the envelope causes a convective zone below the stellar surface which steadily increases in depth.

During these evolutionary phases, both the luminosity and the radius of the star continuously increase, i.e. the star becomes a red giant.

As the luminosity increases, the rate of nuclear burning as well as the rate of the core contraction will increase. Eventually the contraction rate will be so high that the normal thermal equilibrium condition does not apply any further. The contraction will heat the center of the core by

---

*) F. Hoyle and M. Schwarzschild, Ap. J., Suppl. No. 13,(1955)

compression - quite irrespective of the lack of hydrogen fuel for nuclear burning - until it reaches a temperature sufficient for helium burning to commence. This appears to be the critical phase at which the star has reached the top of the observed red giant sequence. When the helium burning starts, it will contribute to the rise in the central temperature. This rise will not, as in the non-degenerate case, be checked by an immediate expansion of the core since the now dominant degenerate pressure is independent of the temperature. Thus an unstable situation may arise for a short evolutionary phase. During this phase, it appears possible that the temperature gradients may become too high to be stable against convection so that convection may set in throughout most of the star, which will mix the helium of the core with the hydrogen of the envelope. Such a thorough mixing would obviously have important consequences for the interpretation of the subsequent evolutionary phases.

So far, the accurate sequence of events during the critical phases at the top of the red giant sequence have not been determined since the onset of helium burning and the active influence of contractive heating make the computations difficult. It is the purpose of this project to investigate these fast and critical evolutionary phases.

21.12 The Basic Equations and Boundary Conditions

The basic conditions throughout the stellar interior are governed by three differential equations. The first one expresses the fact that hydrostatic equilibrium holds, i.e. that the force produced by the gas pressure exactly balances gravity. The second one expresses the relation between the energy flux and the temperature gradient. Here three cases have to be distinguished, depending upon whether the flux is transported by radiation, by convection or by electron conduction, the latter being the case when the gas is degenerate. The third differential equation expresses the conservation of energy; the divergence of the energy flux must be equal to the energy generation by nuclear processes to which, in the phases of fast contraction, the energy freed by non-adiabatic contraction is to be added.

This system of differential equations contains three auxiliary variables: the density, the absorption coefficient, and the rate of energy generation, which are related to the main variables by rather complicated equations.

The density is given as a function of temperature and density and of the abundances of helium and hydrogen. There are two such equations of state, one for the non-degenerate and one for the degenerate case, and a rather involved criterion must be used to decide which equation to apply. The switch is presently made abruptly, while accurately an intermediate equation for incipient degeneracy should be used -- an accuracy which is hardly required here.

The absorption coefficient is given as a sum of two terms. The first represents electron scattering and does not depend on the temperature; the second term combines free-free transitions for hydrogen and helium with bound-free transitions for the heavier elements and is proportional to $T^{-3.5}$. The simple addition of the terms is an approximation, again of sufficient accuracy for the present purpose.

The rate of energy generation is again given as a sum of two terms representing the energy liberated by burning hydrogen and helium, respectively. Probably these two terms will never contribute simultaneously in the same volume since presumably the hydrogen will be completely exhausted before the temperature reaches the threshold for helium burning to set in. Both terms depend, of course, not only upon the temperature but also on the respective abundances of hydrogen and helium which, however, must be replaced by mean values for the convective region where perfect mixing may be assumed.

21.13 Boundary Conditions and Automatic Adjustment of Boundary Values

To complete the definition of the problem the boundary conditions at the center and at the surface of the star have to be assembled.

Our four dependent variables are: the pressure P(r), the temperature T(r), the mass M_r contained in the sphere of radius r, and the luminosity L_r, i.e. the amount of energy radiated per unit time through the surface of that sphere. The boundary conditions at the center, therefore, are

$M_r = L_r = 0$ while P and T are not known a priori. The other two boundary conditions follow from the equilibrium conditions at the surface, i.e. in the photosphere of the star: the first states the relation between the surface temperature and the total luminosity and the second insures that the pressure in the photosphere provides the appropriate optical depth of this layer.

To introduce these two surface conditions in their accurate form would involve appreciable practical difficulties because it would necessitate, first, the addition of the whole set of photospheric equations including partial ionization of hydrogen, and second, the handling of the large ranges in temperature and pressure from the center of the star to its photosphere. These complications are here avoided by the following approximating procedure: The outermost layers of the star are cut off at the point where the temperature reaches exactly one million degrees. This cutoff reduces the total mass only by an entirely negligible amount. It reduces the radius by a somewhat larger percentage, which, however, is still not serious and can be corrected for, if need be, at the end of the calculations. The most serious consequence of this procedure is that the pressure at the cutoff point has to be estimated from previous tentative calculations.

If all of the four boundary conditions were given at the same end of our integration interval, a single numerical integration would yield the configuration at each definite phase. But, unfortunately, this is not the case; we have just seen that we have two conditions at each end. Therefore, a trial and error method must be used to determine the initial values. For example, we could assume certain estimated values for P and T at the center and integrate out to the surface. We could then vary P(o) and T(o) in some systematic way until the computed values would fulfill the boundary conditions at the surface. Or we could start the integrations with estimated values at the surface (fulfilling, of course, the two boundary conditions) and integrate in until, by a proper choice of the two remaining free parameters at the surface, M_r and L_r would come out close enough to zero. --It turned out, however, that neither of these schemes is applicable due to the inherent instability of our system of differential equations as soon as the values of the dependent variables

depart from the physically correct combination. However, a combination of both methods was successful. It basically consists of integrating from both ends towards some intermediate "fitting point" for which the variables can be estimated fairly well. The free parameters at either end are first varied until the values at the fitting point are near the estimated values. The final fitting is then made by linear interpolation.

It is not possible to describe the details of this fitting procedure here, but we should like to mention that Prof. Schwarzschild and Mrs. Selberg succeeded in planning and coding this procedure so that in spite of the highly non-linear character of this "eigenvalue" problem it is carried out by the Computer without human interference except for feeding the initial guesses for the variables at the center, the fitting point and the surface (i.e. the one million degree point).

21.14 General Computational Procedure

The numerical integration of our system of differential equations is hampered by its high degree of non-linearity and the very wide range of almost all variables. Furthermore, the whole system including the various supplementary equations for computing the opacity, energy generation, etc. and the distinction between the radiative, convective and degenerate cases is certainly the most complicated system ever treated on our computer. It is impossible, therefore, to give a full account of the computational procedure in this short report. However, a few remarks may be appropriate here.

To start with, the spatial independent variable is changed from r to M_r. This is essentially a transformation from Eulerian to Lagrangian coordinates, which makes it much easier to follow the changes in chemical composition by nuclear transmutations in each mass element.

Next, let us consider the derivatives with respect to time. Such derivatives occur in the law of energy conservation and in the law for transmutation rates. The energy conservation equation has the character of a heat conduction equation. Therefore, its explicit use for extrapolation in time in a step-wise numerical integration leads to a well known instability except if very small time steps are used, which would be impracticable in

the present application. Consequently, this equation must be used in an "implicit" manner, i.e. when an evolutionary time step leading from previous stellar model to the next stellar model is to be bridged, the equation of energy conservation has to be applied in a symmetrical manner both at the previous time and the new time. On the other hand, the derivatives with respect to time occurring in the equations governing the transmutation rates may be used in an explicit manner for extrapolation in time, no reason being known to suspect here a numerical instability. The most simple linear extrapolation formula would be sufficient for the accuracy required here, but it may lead, on occasion, to erroneous negative values for the hydrogen or helium abundance. A suitable correction was used to avoid this condition.

Altogether the handling of one evolutionary time step is thus reduced to the following computation: after the derivation of the configuration for one definite time has been completed, the contraction and the new hydrogen and helium abundances are computed for every point in the star. With these three functions the time interval is bridged and the configuration at the next phase can be derived.

To adjust all the equations to the requirements of the electronic computer, the physical variables had to be scaled appropriately. Most of them were scaled linearly i.e. multiplied by an appropriate power of 2 or/and 10; in addition the density ρ and the pressure P were replaced by their third and fourth roots, respectively, in order to reduce the range of the variables handled by the computer. Furthermore, the distance r from the center was replaced by the variable $h = 1/r$, which has its highest number of significant figures in the important central region of the star rather than in the less important envelope. The time step was scaled in such a way as to insure approximate constancy of the amount of hydrogen or helium burned per time step.

The introduction of the variable $h = 1/r$ leads to a (non-essential) singularity at the center of the star. Therefore, a simple analytic approximation is used to start the numerical integration at some small, but finite radius. From thereon the length of each step is computed by the machine from the requirement that no variable must change by more than some fixed fraction of its value.

21.15 Machine Utilization

The length and complexity of the code for the present computation confronted us with several problems which are not normally encountered in smaller calculations. We shall discuss some of them in this paragraph.

One of the most difficult ones was the problem of storage. Since it was not possible to accommodate the entire code, including all the data to be stored for each model, in the Williams memory (1924 words), several subroutines were stored on the magnetic drum and called into the Williams memory whenever needed. This procedure proved to be not only very time consuming -- especially with the old drum which had an access time of nearly 200 msec. -- but it also would have required a reliability of the magnetic drum and its amplifying circuitry that could not be achieved with the old drum. Both considerations hold, to a lesser degree, even for the new drum. It was tried, therefore, to reduce the number of drum references by accommodating in the Williams memory at least those subroutines which are most frequently needed, i.e. the ones occurring in every integration step.

When the new drum became available, a restart procedure was added to the old code. Storing the code and the data (about 3000 words) at the end of each full integration (i.e. from center to surface), it was possible, in case of obvious machine trouble, to resume the computation at the end of the previous integration. In addition, an optional automatic duplication mechanism was recently incorporated in our code.

As mentioned in the previous paragraph, scaling presented a considerable problem. In fact, a fixed point machine is not very well suited to this type of problem, in which the variables vary over a very wide range. It requires careful and accurate scaling in order to preserve a sufficient degree of accuracy; moreover, since the ranges of the variables may differ from model to model, the scaling has to be changed accordingly. Two possibilities were investigated to solve this difficulty: (i) the use of coded floating point techniques, and (ii) the use of logarithmic variables. Of the two, the latter seems more adequate since our formulae contain a considerable number of integer and fractional powers. It is planned, therefore,

to rewrite the code if the subroutines for log x, exp x, and log (x + y) can be made fast enough and the Williams memory can accommodate them during a whole integration step. This would also solve the problem of read-around in the Williams memory, which arose in the evaluation of fractional powers and could only be overcome by an artificial slow-down.

So far we have computed nine models. The average computing time for one model is 3+1/2 hours. Actually, many more machine hours were spent on debugging and on experimentation with new computational methods, and a good many were lost due to machine errors.

21.16 Results and Outlook

With the procedure described above a sequence of nine models has so far been computed which, for a specific star, covers a range of advanced evolutionary phases in which the star has red giant character, in satisfactory agreement with astronomical observations. The results are in general agreement with previous investigations,[1] but they indicate that the contraction term in the third basic equation plays a more important role for the structure of the advanced evolutionary phases than the earlier rough estimates had indicated.

It is planned to continue the present evolutionary model sequence up to and beyond the phase when helium burning sets in, and to carry through other evolutionary sequences for stars of different masses and initial compositions, for the purpose of obtaining a secure basis for the theory of those most advanced evolutionary phases in which the formation of the heavy elements is expected to occur.

---

1) cf. Numerical Integrations for the Stellar Interior, Härm, R. and Schwarzschild, M., Astrophysical J., 121 (1955) and Astrophysical J., Suppl. Series Vol. I, p. 319-430 (1955), where numerous further references are given.

It is also planned to prepare technical reports and/or papers for a scientific journal[1] whenever an important phase of this problem has been finished.

The supporting contracts for this work are: Contract No. Da-36-034-ORD-1646 for machine operation, Contract No. Nonr 1358-(03) for all coding and, while this report is being written, Contract Nonr 1358-(04) for further use of the machine. (January 1, 1957 - June 30, 1956.)

1) presumably the Astrophysical Journal and/or its Supplementary Series.

21.20 ADIABATIC PULSATIONS OF AN ORIGINALLY ISOTHERMAL ATMOSPHERE

Originator and Coder: D. A. Lautmann[1)]

The periodic variation of brightness of certain stars, named Cepheids after the most famous of them, viz. Cephei, is one of the most intriguing problems of astrophysics. The so-called pulsation theory attempts to explain these variations on the basis of hydrodynamic pulsations.[2)] The theory of these pulsations is fairly well understood for the linear case, i.e. as long as the amplitudes of the oscillations are so small that quadratic terms can be neglected. In the deep interior of a star, where energy considerations demand small amplitudes, the highly developed linear theory of adiabatic pulsations has been applied and is in fairly good agreement with observation. In the atmosphere of a variable star, however, amplitudes become large and non-linear terms important. This is confirmed by the fact that the observed variations of brightness are not sinusoidal functions of time but show a pronounced skewness. It was therefore decided to find numerical solutions for the original non-linear equations of hydrodynamics but with the following simplifying assumptions:

(i) The Cepheid atmosphere is considered to be plane-parallel.

(ii) The atmosphere is originally isothermal and in hydrostatic equilibrium under constant gravity.

(iii) All changes are adiabatic.

(iv) The influence of the interior of the star is approximated by a sinusoidal motion of the bottom layer of the atmosphere.

---

1) cf. Lautman, Don A., Doctoral Thesis, Princeton University, 1956. To be published in Astrophys. J., 1957.

2) cf. Eddington, A. S., The Internal Constitution of Stars, chapter viii. Cambridge University Press, 1926.

The formulation of boundary conditions for the top of the atmosphere is a much more difficult problem. An isothermal atmosphere theoretically extends to infinity, but in a numerical computation it clearly must be cut off at a certain point. The upper boundary condition should, therefore, express the effect of the neglected part of the atmosphere.

It is essential for this investigation that the oscillation of the bottom layer generates outgoing traveling waves of higher frequency which are not reflected at the "surface" of the atmosphere. The upper boundary condition was constructed to fulfill this requirement.

21.21 Basic Equations of the Problem

We introduce the following symbols:

v	velocity	r	position	r_o	}	same as r, ρ, p,
t	time	ρ	density	ρ_o	}	but for hydrostatic
g	gravity	p	pressure	p_o	}	equilibrium

The time t and the "initial" position r_o are used as independent variables. The four basic equations then read:

Equation of motion
$$\frac{\partial v}{\partial t} = -g - \frac{1}{\rho_o}\frac{\partial p}{\partial r_o} \tag{1}$$

Hydrostatic equilibrium
$$\frac{\partial p_o}{\partial r_o} = -\rho_o g \tag{2}$$

Equation of continuity
$$\frac{dr}{dr_o} = \left(\frac{\rho}{\rho_o}\right)^{-1} \tag{3}$$

Adiabatic equation
$$\frac{p}{p_o} = \left(\frac{\rho}{\rho_o}\right)^{\gamma} \tag{4}$$

If p/p_o is used as variable, eq. (1) can be written in the form

$$\frac{\partial v}{\partial t} = -\frac{\rho_o}{\rho}\cdot\frac{\partial}{\partial r_o}\left(\frac{p}{p_o}\right) + g\left(\frac{p}{p_o} - 1\right) \tag{5}$$

where eq. (2) has been used to eliminate dp_o/dr_o. This transformation would simplify the computation since implicit use of eq.(2) eliminates the necessity to compute the exponential function for p_o and ρ_o. It also avoids certain scaling difficulties on a fixed-point computer.

The boundary condition at the bottom is simply

$$v = A\sin\omega t \quad . \tag{6}$$

A suitable boundary condition for the top can be derived from the analytic solution of the linearized equations.

21.22 Analytic Solution of the Linearized Equations

Let us put $y = r - r_o$ for the displacement from the position of static equilibrium, hence $v = \partial r/\partial t = \partial y/\partial t$. From equations (3) and (4) we obtain

$$\frac{\partial r}{\partial r_o} = \left(\frac{p}{p_o}\right)^{-1/\gamma} = \left(1 + \frac{p-p_o}{p_o}\right)^{-1/\gamma}$$

We now use the assumption that the oscillations have small amplitudes, thus $|p-p_o| \ll p_o$ and

$$\frac{\partial r}{\partial r_o} \approx 1 - \frac{1}{\gamma}\cdot\frac{p-p_o}{p}$$

or

$$\frac{p}{p_o} \approx 1 - \gamma\cdot\frac{\partial y}{\partial r_o}$$

This brings eq.(5) into the linear form

$$\frac{\partial^2 y}{\partial t^2} = \gamma c_o^2\cdot\frac{\partial^2 y}{\partial r_o^2} - \gamma g\cdot\frac{\partial y}{\partial r_o} \quad , \tag{7}$$

where $c_o = p_o/\rho_o$ is a constant for ideal gases. Assuming a time dependence of the form $\exp(i\Omega t)$, the solution will greatly depend on the frequency Ω. If Ω is very high, the wave length is very short and the gravitational term can be neglected. These short waves are, therefore, traveling with the speed of sound, viz.

$$u = c_o\gamma^{1/2} \quad .$$

For a next approximation let us put

$$y = \exp(ikr_o - i\Omega t)$$

hence $\partial / \partial r_o = ik$ and $\partial / \partial t = -i\Omega$. Introducing these expressions in (7) we get

$$(-i\Omega)^2 = \gamma c_o^2 (ik)^2 - \gamma g ik = \gamma (c_o ik)^2 \left(1 - \frac{g}{c_o^2 ik}\right)$$

or, if $k \gg g/c_o^2$

$$-i\Omega \approx \sqrt{\gamma}\, c_o ik \left(1 - \frac{g}{2c_o^2 ik}\right) = \sqrt{\gamma}\left(c_o ik - \frac{g}{2c_o}\right)$$

Replacing $i\Omega$ and ik by the corresponding operators we get

$$\frac{\partial y}{\partial r_o} \approx \frac{gy}{2c_o^2} - \frac{1}{c_o \gamma^{1/2}} \cdot \frac{\partial y}{\partial t} \quad , \qquad (8)$$

which was used as upper boundary condition for most cases. As a variation, a slightly different boundary condition was used to represent a high temperature layer (corona) at the top of the atmosphere.

21.23 Computational Procedure

As long as the linearized equations can be used, the harmonic perturbation at the bottom layer will generate a harmonic movement of the whole atmosphere. In the general, i.e. non-linear case, this movement will still be periodic, but not harmonic. It was felt that the periodicity could not be used efficiently to simplify the problem. The time t will, therefore, appear explicitely as an independent variable together with r_o.

The computation was started from the hydrostatic equilibrium, i.e. $r = r_o$ and $v = 0$ with an oscillation (cf. eq. 6) at the bottom. The integration of the differential equations was continued until the motion of the whole atmosphere became essentially periodic.

In order to perform the numerical computations, the (r_o, t) plane is divided into a grid of constant spacing Δr_o and Δt. To obtain the highest accuracy from simple, low-order approximations it is necessary to use centered differences. This requires that the variables be carried on staggered points of the grid according to the following list:

VARIABLES	SPACE STEP	TIME STEP
$y,\ \partial^2 y/\partial t^2$	integer	integer
$v = \partial y/\partial t$	integer	half-integer
$\partial y/\partial r_o\ ,\ p\ ,\ \varrho$	half-integer	integer

All variables are then replaced by dimensionless variables such as $w = v/c_o$ and $\tau = tg/c_o$ for speed and time, respectively, and all differentials are replaced by simple central differences.

The computational procedure is as follows: Suppose we know all relevant variables, including $\partial v/\partial t$, for a certain time t and that our time step equals Δt. First $v(r_o)$ is computed for all r_o's from

$$v(t-\tfrac{\Delta t}{2}) + \Delta t.\ \frac{\partial v}{\partial t}\ (t) \Longrightarrow v(t+\tfrac{\Delta t}{2})\ ,$$

except at the boundaries where eq.(6) and (7) are used. Similarly

$$y(t) + \Delta t.\ v(t+\tfrac{\Delta t}{2}) \Longrightarrow y\ (t+\Delta t)$$

Finally, the finite-difference equivalents of eqs. (3), (4), and (1) or (5) are used, in this sequence, to compute ϱ, p and $\partial v/\partial t$ for $t + \Delta t$, whereby the cycle is completed.

One computational detail may be worth mentioning: In eq.(4) ϱ/ϱ_o must be raised to the power γ^-, which in this computation was taken as 5/3. Originally, an iterative method with constant initial "guess" was used to compute the cube root, but it was soon discovered that this took almost half of the total computation time. It was replaced, therefore, by a simple rational approximation, viz.

$$(Az + B)\cdot z + C \Longrightarrow z^{5/3}$$

for each of three ranges of z, yielding an accuracy of about 1%.

21.24 Numerical Stability

When approximating a differential equation by a difference equation we expect certain errors in the solution. One type of error (truncation error) arises from the neglect of higher terms in the series expansions which replace the derivatives. The truncation error can usually be made as small as one wants

by either using a higher order approximation for the derivatives or by reducing the grid spacing. Another type of error is due to the fact that the difference equation may not be stable even though the differential equation is.

In order to avoid this kind of instability, an elaborate analysis was carried out for which, however, our system of equations had to be linearized. The results of this analysis hold strictly only for the linearized system but they give a fair indication of what one might expect from the non-linear equations.

The result from the linearized system is that the Courant-Friedrich condition[1]

$$\frac{\Delta r_o}{\Delta t} > u \qquad = \text{velocity of sound}$$

must be fulfilled which, for a given space step, sets an upper limit for the time step. But because the atmosphere will be compressed during part of the pulsation cycle and the velocity of sound, therefore, increased, it becomes necessary to make Δt smaller than $\Delta r_o/u$.

We previously mentioned the advantages gained in simplicity and speed of computation if eq.(1) is replaced by eq.(5). In actual computation, however, it is found that the numerical solutions of this modified system suffer from a very strange type of instability. The errors increase slowly to a fairly constant value which, though small, is big enough to destroy the periodicity of the solutions. A possible explanation is given in Lautman's Thesis[2] to which we refer the interested reader.

21.25 Results

The numerical integration was carried out for four different frequencies and for two different amplitudes at the bottom, while the height of the atmosphere was generally taken as 10 "scale-heights", $H = c_o^2/g$. In addition to these eight

---

1) Courant R, Friedrichs K. and Lewy H. , Math.Ann. 100; page 132,(1928)

2) Lautman, Don A., Doctoral Thesis, Princeton University, 1956

runs, several experiments with varying parameters (height, time and/or space steps) were carried out, and also one involving a corona. Each run took about one hour on the machine.

The first period will always show strong transients which soon settle down; the motion is essentially periodic by the sixth period.

The results show that, even with the simple model which we have chosen, it is possible to account for the observed skewness of the velocity curve (as a function of time) and for certain humps on the descending branch. Both of these effects are very strongly dependent on the boundary condition, however, and in further work on this subject it will be necessary to analyse in much more detail the effect of the upper parts of the atmosphere on the character of the pulsation. The travelling wave components which lead to the phase lag are seen to arise quite simply from the standing wave fundamental of the interior, but the magnitude of the lag is slightly less than that observed. It should again be emphasized that the simplifying assumptions which have been made -- notably the adiabatic approximation and the plane-parallel atmosphere -- probably influence the final results greatly.

21.30 MAGNETOHYDRODYNAMIC THEORY OF SOLAR SPICULES

Originator and Coder: Reimar Lüst

During the last two years, calculations concerning this problem have been carried out by Dr. Schlüter, Dr. Schwarzschild and Mr. Lautman of the Princeton University Observatory. They have calculated pressure waves which expand in an isothermal atmosphere from the base of the chromosphere in the presence of a vertical magnetic field. An essential simplification for this problem was the assumption of waves with small amplitudes which allowed the basic differential equations to be linearized.[1)]

The aim of the present investigation is to calculate waves of arbitrary amplitude. Therefore, the non-linear differential equations have to be integrated. In the equations of motion an artificial viscosity term was introduced in analogy to the method of von Neumann and Richtmyer[2)] in order to allow for the possibility of considering the transition of a wave with finite amplitude into a shock wave.

21.31 Basic Equations

The basic equations for this problem are the following:

Momentum equation:

(1) $$\rho \frac{d\vec{v}}{dt} = \vec{j} \times \vec{H} - \operatorname{grad}(P+q) - \rho \operatorname{grad}\Phi$$

Equation of continuity:

(2) $$\frac{\partial \rho}{\partial t} = -\operatorname{div}(\rho \vec{v})$$

Energy equation:

(3) $$\frac{dP}{dt} = \left[\gamma \frac{P}{\rho} + (\gamma - 1) \frac{q}{\rho} \right] \frac{d\rho}{dt}$$

Ohm's law (with $\sigma = \infty$):

(4) $$\vec{E} = -\vec{v} \times \vec{H}$$

---

1) For detailed description of the astrophysical problem of spicules see Final Report on Contract No. DA-36-034-ORD-1330, II,56 (1954).

2) von Neumann, J. and Richtmyer, R. D., A method for the numerical calculation of hydrodynamic shocks. J. Appl.Phys. 21, 232-237 (1950).

Maxwell's equations:

(5a) $$\nabla \times \vec{E} = -\frac{\partial H}{\partial t}$$

(5b) $$\nabla \times \vec{H} = 4\pi \vec{j} \qquad \left(\text{as } \left|\frac{\partial \vec{D}}{\partial t}\right| \ll |\vec{j}|\right)$$

where

ρ = density, p = pressure, v = velocity, Φ = gravity potential,
E = electric field, H = magnetic field, j = current density,
γ = ratio of the specific heats, t = time.
q is an artificial viscosity term and is given by the following expression:

$$q = \frac{\beta}{\rho}(\Delta x)^2 \frac{d\rho}{dt}\left|\frac{d\rho}{dt}\right| = -\beta(\Delta x)^2 \rho \, \mathrm{div}\,\vec{v}\,|\mathrm{div}\,\vec{v}| \quad ;$$

β is a constant, and Δx is the interval length in space used in the numerical computation.

21.32 Assumptions

The same assumptions as in the previous investigation have been made, namely,

a) plane parallel chromosphere,
b) barometric density gradient ($\rho \sim e^{-az}$, $a = g\rho/p_o = \text{const.}$)
c) small vertical extent ($\nabla\Phi = g = \text{const.}$),
d) axial symmetry ($H_\psi = v_\psi = 0$; $E_z = j_z = j_s = 0$),
e) homogeneous vertical magnetic field of value H_o,
f) all changes adiabatic, apart from the artificial viscosity term.

21.33 Axially-Symmetrical and Non-Dimensional Equations

With the assumption of axial symmetry all independent variables should be only a function of the two space variables s (distance from the vertical axis of symmetry) and z (distance from the horizontal base plane) and of the time t. The velocity $\vec{v}$ and the magnetic field $\vec{H}$ are given by the horizontal components v_s and H_s respectively, and by the vertical components v_z and H_z.

The following non-dimensional variables are introduced:

$$r = \frac{g}{c^2} s \quad ; \quad y = \frac{g}{c^2} z \quad ; \quad \tau = \frac{g}{c} t \quad ; \quad c^2 = \frac{p_o}{\rho_o} = \text{const.}$$

$$w = \frac{v_s}{c} \quad ; \quad v = \frac{v_z}{c} \quad ; \quad \rho' = \frac{\rho}{\rho_o} \quad ; \quad p' = \frac{p}{p_o} \quad ; \quad q' = \frac{q}{q_o}$$

$$S = \frac{H_s}{H_o} \quad ; \quad Z = \frac{H_z}{H_o} \quad ; \quad B = \frac{H_o^2}{c^2 \rho_o} = B_o e^{y}$$

ρ_o = undisturbed density; p_o = undisturbed pressure.

With these variables one gets the following equations after eliminating the electric field $\vec{E}$ and the current density $\vec{j}$: and writing ρ, p, q for ρ', p', q':

$$\text{(6a)} \quad \rho \frac{\partial w}{\partial \tau} = -w \frac{\partial w}{\partial r} - v \frac{\partial w}{\partial y} - Z \cdot B \cdot \left(\frac{\partial Z}{\partial r} - \frac{\partial S}{\partial y} \right) - \frac{\partial (p+q)}{\partial r}$$

$$\text{(6b)} \quad \rho \frac{\partial v}{\partial \tau} = -w \frac{\partial v}{\partial r} - v \frac{\partial v}{\partial y} + S \cdot B \cdot \left(\frac{\partial Z}{\partial r} - \frac{\partial S}{\partial y} \right) - \frac{\partial (p+q)}{\partial y} + P + q - \rho$$

$$\text{(7)} \quad \frac{\partial \rho}{\partial \tau} = -w \frac{\partial \rho}{\partial r} - v \frac{\partial \rho}{\partial y} - \rho \left(\frac{w}{r} + \frac{\partial w}{\partial r} - v + \frac{\partial v}{\partial y} \right)$$

$$\text{(8)} \quad \frac{\partial p}{\partial \tau} = -w \frac{\partial p}{\partial r} - v \frac{\partial p}{\partial y} - \left[\gamma P + (\gamma - 1) q \right] \left(\frac{w}{r} + \frac{\partial w}{\partial r} + \frac{\partial v}{\partial y} \right) + P v$$

$$\text{(9a)} \quad \frac{\partial S}{\partial \tau} = -v \frac{\partial S}{\partial y} - S \frac{\partial v}{\partial y} + w \frac{\partial Z}{\partial y} + Z \frac{\partial w}{\partial y}$$

$$\text{(9b)} \quad \frac{\partial Z}{\partial \tau} = -w \frac{\partial Z}{\partial r} - Z \left(\frac{w}{r} + \frac{\partial w}{\partial r} \right) + v \frac{\partial S}{\partial r} + S \left(\frac{v}{r} + \frac{\partial v}{\partial r} \right)$$

$$\text{(10)} \quad q = -\beta (\Delta r)^2 \rho \left(\frac{w}{r} + \frac{\partial w}{\partial r} + \frac{\partial v}{\partial y} \right) \cdot \left| \frac{w}{r} + \frac{\partial w}{\partial r} + \frac{\partial v}{\partial y} \right|$$

21.34 Boundary and Initial Conditions:

To simplify the problem it is assumed that the wave is travelling in a "box". Therefore, the normal components of the velocity shall vanish at the boundaries. For the top the assumption is made that both components of the velocity are zero in order to avoid incoming waves. Besides, boundary conditions for the magnetic field have to be introduced. As a simplification it is assumed that the magnetic field is always perpendicular to the bottom, and parallel to the vertical boundaries.

As initial condition a "pressure hump" is assumed which should start from the origin. The initial values for the 6 variables are then given by

$$w = v = 0$$
$$p = \varrho = 1 + C.\exp\left[-\alpha^*(r^2+y^2)\right]$$
$$S = 0$$
$$Z = 1.$$

Here, C gives the amplitude of the pressure hump and α^* is inverse proportional to the decay length.

21.35 Difference Equations

To convert the differential equations (6a) - (10) into difference equations, the space differential quotients are replaced by difference quotients correct to the second order, while the time differential quotients are replaced by difference quotients correct to the first order; r, y, and τ are introduced as independent variables.

$$r = R\cdot\Delta r \qquad 0 \le R \le \mathcal{R}$$
$$y = Y\cdot\Delta y \qquad 0 \le Y \le \mathcal{Y}$$
$$\tau = T\cdot\Delta\tau$$

The differential quotients are then given by

$$\left.\frac{\partial w}{\partial r}\right|_{R,Y} = \frac{1}{2\Delta r}\left(W_{R+1,Y} - W_{R-1,Y}\right) \quad ; \quad \left.\frac{\partial w}{\partial y}\right|_{R,Y} = \frac{1}{2\Delta y}\left(W_{R,Y+1} - W_{R,Y-1}\right)$$

$$W_{R,Y}^{T+1} = W_{R,Y}^{T} + \Delta\tau\cdot\left.\frac{\partial w}{\partial \tau}\right|_{R,Y}^{T}$$

and analogous for the other variables.

21.36 Coding and Numerical Computations

The problem could not have been treated without the aid of the new large magnetic drum. With this appreciably enlarged storage it was possible to increase the number of points in the grid, which was chosen to be 19 x 30 points, ($\mathcal{R}$ = 19, $\mathcal{Y}$ = 30). On the drum, the values of the 6 variables w, v, ϱ, p, S, and Z for each point from the preceding time step T were stored. Further, the values of the next time step T + 1 were stored at those storage places where the values of the time step T - 1 had been before. The values for one point were stored in successive places. The values for three lines of the grid, namely for R - 1, R, and R + 1 are needed for the calculation of the line R for the next

time step. Since the Williams memory was not large enough to store all the values of three lines, each line was split into three parts which were separately shifted from the drum to the Williams memory and back.

The whole code was also stored on the drum because the Williams memory was too small. The code for the six difference equations was divided into three parts: part one for the calculation of the values at the bottom, part two for the main field including the two vertical boundaries, and part three for the top. Further parts of the code contained the calculation of the initial values and the orders for punching out the results.

For reasons of checking every time step was calculated twice. The calculations were only carried on if both results were in agreement. If the results differed, the machine stopped to provide information for the operator to the effect that an error had been made. Restarting caused the time step to be repeated again. The integration of one time step for all the points of the grid took about two minutes. The magnitude of a time interval determined how often the results were punched out, and the values for every second point of the grid were punched. The time needed for punching out the values for one time step was about 4 1/2 minutes. Extensive restart procedures were incorporated into the code so that, in principle, no more than about 10 minutes of computing time would be lost due to machine errors.

21.37 Results

So far, for two sets of parameters 140 time steps have been calculated. In the first case, the amplitude of the pressure hump was small ($C = 0.01$) and resulted in waves with small amplitudes. In the second case, the amplitude of the pressure hump was equal to the initial pressure ($C = 1$) and led to waves with much larger amplitudes. In both cases, the strength of the magnetic field was chosen in such a way that the magnetic pressure was equal to the gas pressure in the middle between the bottom and the top. Though according to the computations which have been carried out until now the wave has not yet reached the top of the grid, one can see the influence of the magnetic field. The magnetic field has the effect that the waves are strongly guided into the vertical direction.

21.40 AN IMPROVED SOLAR MODEL WITH THE CARBON CYCLE INCLUDED

Originator and Coder: Ray Weymann

A recent model of the solar interior by Schwarzschild, Howard, and Härm [1] in which all energy was generated by the proton-proton reaction, appropriate for dwarf stars, indicated that the central temperature was approaching the point where the carbon-cycle (which provides the energy for more massive stars) would contribute significantly to the total amount of energy generated. In the present study, the effect of the carbon cycle was taken into account. In addition, an improved distribution of the variable chemical composition of the sun was used.

The equations involved are those of mechanical and thermal equilibrium, with two-point boundary conditions, determining two eigenvalues. These are the first-order non-linear ordinary differential equations, whose form varies in different parts of the star. In the outer portion, the envelope, the equations are those appropriate to convective equilibrium (cf. [2]) and Section 21.5 of this report, eq. (1)), while in the central region, the equation for the temperature distribution is that appropriate for radiative equilibrium. Since the envelopes had been previously integrated, it was now necessary to integrate the equations for the central region and fit the two solutions together at the point at which convection sets in.

The entire series of integrations, including computations of the starting values (which must be done by series, due to the existence of singularities in the coefficients at the center) was performed on the computer, using the FLINT code, and was fitted to the envelopes by hand. The results for this improved solar model differ only slightly from those obtained previously.

---

1) Schwarzschild, Howard, and Härm, Astrophys. J., 125, 233(1957).

2) Osterbrock, Astrophys. J., 118, 529(1953).

21.50 ON THE PULSATIONAL STABILITY OF STARS WITH CONVECTIVE ENVELOPES

Originator and Coder: Irving N. Rabinowitz

In order to study the possibility of cepheid pulsations being maintained by thermonuclear energy generation in the deep interior of the stars, it is necessary, first, to construct models of cepheid variables, and second, to solve the pulsation equations arising from a perturbation of the model. In this investigation, three models were constructed which differed from those studied by previous investigators in that they had convective envelopes rather than radiative ones. Such envelopes are described by the equations[1]

$$\frac{dM_r}{dr} = 4\pi r^2 \rho \quad , \quad \frac{dP}{dr} = -\frac{GM_r}{r^2}\rho \quad , \quad P = KT^{2.5} \tag{1}$$

These are the equations for conservation of mass, mechanical equilibrium, and convective equilibrium, respectively. Integrations of these equations had been previously performed by Härm and Schwarzschild[2] on the IAS computer, but not to depths great enough for the present study. It was therefore necessary to extend the integrations for the three envelopes which were used.

The wave equation derived from linearizing the general perturbation equation was broken down into two first-order equations for the radius change and the density change, respectively:

$$\begin{aligned} &\frac{dr'}{dy} + \rho' + 3r' = 0 \\ &\frac{d\rho'}{dy} - V(y)\rho' - V(y)\left[2.4 + \frac{x^3}{q}\omega^2\right] r' = 0 \end{aligned} \tag{2}$$

where $y = \log x$.

The quantity ω^2 is an eigenvalue representing the period of pulsation, and these equations have well-behaved solutions for only discrete values of ω^2.

---

1) cf. Osterbrock, Astrophys. J., 118, 529(1953).

2) Härm and Schwarzschild, Astrophys. J.Suppl. Series, 1, 319(1955).

21.51 Methods of Computation

The method of determination of the eigenvalue was as follows: several integrations of (2) for various values of ω^2 were done on the machine and plotted by hand. From the graphs, it was possible to choose another set of ω^2 values more closely approximating the desired value. This process converges quite rapidly, since by interpolation in the graphs it is possible to guess at the eigenvalue with perhaps one-and-a-half decimal place accuracy in the differences, so that no more than six runs on the machine, each requiring about an hour, were necessary to determine the eigenvalue to nine places. This procedure was carried out separately for the three models in question.

The method of integration used was an extremely simple one, namely Heun's method, in which the value of the function at the forward point is determined by the trapezoidal rule, and is then improved by taking the average of the two slopes and using this to determine a new value at the forward point. This is a primitive type of Runge-Kutta method, of second-order accuracy and rather high speed. It may be remarked that the small accuracy of the integration method does not conflict with the necessity of finding the eigenvalue to full accuracy, the reason being that it is the shape of the density-change curve that counts, rather than the value of ω^2. Thus, changing the truncation error by using a finer grid, or a more accurate integration method has the effect of changing the value of ω^2 to which the well-behaved solution belongs, but has practically no effect upon the solution itself.

All computations were performed using the floating-point interpreter (section 12), and took a total of about twenty hours on the machine. A single integration, including punching out a table of the results, took about four minutes.

21.52 Results

It was found that even for models with convective envelopes the energy generation by thermonuclear processes in the deep interior was insufficient to maintain the pulsations, since dissipative processes in the envelope lose more energy than is generated as a result of the pulsations. The same property has been previously found in models with radiative envelopes. Since the deep interior is incapable of explaining the long-time stability of cepheid variables, it now becomes necessary to study the mechanisms of energy storage and dissipation in the envelope.

22. ATOMIC AND NUCLEAR PHYSICS

22.10 THE GROUND STATE OF THE HELIUM ATOM

Originator: T. Kinoshita

Coder: Mrs. S. Bargmann

Prof. Toichiro Kinoshita has carried out an elaborate and very accurate recomputation of the ground state of the Helium atom[1] in order to match the improved accuracy of measurement of this energy level.

It is well known that this three-body problem (nucleus + 2 electrons) cannot be solved analytically. Very close upper bounds can be computed, however, by help of the RITZ variation method if a sufficiently large set of suitably chosen coordinate functions is used. Two small corrections -- one for the mass polarization of the nucleus, the other for relativistic terms -- may be estimated with sufficient accuracy after the unrelativistic three-body problem has been solved.

Prof. Kinoshita used more general, and obviously more suitable, coordinate functions than his predecessors and he finally increased the number of coordinate functions to 39. The major part of this huge numerical work was carried out on the AEC-UNIVAC at New York University. Preliminary computations, with up to 10 functions, have been done on our machine in Princeton. We wish to give a short description of the mathematical problem and the method used for its solution.

22.11 Basic Equations

The original quantum-mechanical problem consists in minimizing the integral

$$E = \int \psi (H\psi)\, d\omega$$

under the normalization condition

$$N = \int \psi^2 d\omega = 1$$

Here, ψ is the (real) wave function of the Helium atom which depends on the 6 coordinates of the two electrons; H is the Hamiltonian operator, and $d\omega$

---

1) Kinoshita, T. Phys. Rev. 105, 1490 (1957)

indicates integration over the 6 coordinates. Three angular variables are separated so that in effect only three independent variables remain.

If ψ is expressed as a linear combination of trial functions χ_i with coefficients u_i to be determined,

$$\psi = \sum_{i=1}^{n} u_i \chi_i \quad , \qquad (1)$$

the two integrals E and N are turned into quadratic forms in the coefficients u_i:

$$E' = \sum_{i,j} \alpha_{ij} u_i u_j = (u, Au)$$

$$N' = \sum_{i,j} \beta_{ij} u_i u_j = (u, Bu) \quad .$$

(u is the vector with components u_i, A and B are symmetric matrices with elements α_{ij} and β_{ij} respectively, and B is positive definite).

The original problem is then replaced by that of minimizing E' under the subsidiary condition N' = 1, which in turn leads to the eigenvalue problem

$$Au = \lambda Bu \qquad (2)$$

The lowest eigenvalue λ is the minimum of E', and for an appropriate choice of the trial functions χ_i it furnishes an approximate value (in fact, an upper bound) for the minimum of E. With the help of eq.(1) the corresponding eigenvector furnishes an approximate wave function.

In our computations, n was equal to 10.

We determined all the eigenvalues of (2), not merely the lowest one, and all eigenvectors.

22.12 Computational Procedure

Although the algorithm used here has been described in the Final Report on Contract No. Da-36-034-ORD-1330, December 1954, we shall shortly indicate it here for the reader's convenience. The main steps are:

1) Diagonalization of B, i.e. determination of an orthogonal matrix U such that

D = U*BU

is a diagonal matrix (with positive diagonal elements δ_i). Here, U* is the transpose of U.

2) Formation of $D^{-1/2}$, the diagonal matrix with elements $\delta_i^{-1/2}$.

3) Formation of the matrix product $UD^{-1/2}$.

4) Formation of the symmetric matrix

$$W = (UD^{-1/2})^* A (UD^{-1/2})$$

5) Diagonalization of W in the form

$$\Lambda = V^* W V$$

where V is orthogonal, and Λ is a diagonal matrix with elements λ_i.

6) Formation of the matrix product

$$Z = (UD^{-1/2})V \quad .$$

Then the λ_i are the eigenvalues, and the columns z_i of the matrix Z are the eigenvectors of the problem (2)

To prove this, we first note that

$$Z^* A Z = V^* (UD^{-1/2})^* A (UD^{-1/2}) V = V^* W V = \Lambda$$
$$Z^* B Z = V^* D^{-1/2} U^* B U D^{-1/2} V = V^* D^{-1/2} D D^{-1/2} V = I$$

Setting $u = Zy$, and premultiplying both sides of (2) by Z^* we obtain the equivalent eigenvalue problem

$$\Lambda y = \lambda y$$

in terms of the vector y. Since Λ is diagonal, its eigenvalues are λ_i, the corresponding eigenvectors e_i (e_i has the i-th component 1, and all other components 0). Thus the eigenvectors of (2) are $Ze_i = z_i$, q.e.d.

The algorithm described here requires two diagonalizations which are carried out according to the Jacobi method (see Final Report on Contract No. DA-36-034-ORD-1023, p.II-50, where also the checking procedures are described.).

We succeeded in somewhat improving the accuracy obtained before by increasing the number of rotations and by testing explicitly for the best scaling factor at intermediate stages of the computation, readjusting our numbers accordingly.

22.20 RELATION BETWEEN THE VIBRATION FREQUENCIES OF A CRYSTAL AND THE SCATTERING OF SLOW NEUTRONS

Originator: G. L. Squires

Analyst and Coder: Hans J. Maehly

Under this title Dr. G. L. Squires, has published a paper*) reporting and discussing the results of computations which were, according to his specifications, carried out on the Electronic Computer at the Institute for Advanced Study. We wish to supplement this paper by a purely mathematical description of the problem and an outline of a few interesting aspects of our code.

22.21 The Computational Problem

The entire problem can readily be divided into 3 parts.

(i) Computation of the Frequencies

Given 5 constants a, b, c, e, f, compute the eigenvalues λ_m and "frequencies" $\mu_m = \sqrt{\lambda_m}$ for which the 3x3 determinant

$$| T(a) - \mu^2 I | = 0$$

will vanish, where I is the unit matrix and

$$T_{ii} = a + 2b - aC_jC_k - bC_i(C_j + C_k) + eS_i^2 + f(S_j^2 + S_k^2)$$

$$T_{jk} = cS_jS_k \quad , \quad i \neq j \neq k$$

$$\left.\begin{aligned} S_i &= \sin\alpha_i \\ C_i &= \cos\alpha_i \end{aligned}\right\} \quad \begin{aligned} &i = 1(1)\ 3 \\ &\alpha_i = 0(\delta)\ \pi - \delta \end{aligned}$$

*) Phys. Rev. 103, 304-312 (1956).

By making full use of the various symmetries, the number of grid points $(\alpha_1, \alpha_2, \alpha_3)$ can be reduced considerably, namely from 1000 to 152 for $\delta = 18^\circ$ and from 27.000 to 2792 for $\delta = 6^\circ$. These are the values of δ for which the computation was actually carried out.

(ii) Computation of the Frequency Distribution Function

After arranging the frequencies $\mu_m(\alpha_1, \alpha_2, \alpha_3)$ according to their size, determine the number N_i of such frequencies lying between μ_i and $\mu_i + \Delta\mu$ for some given and finite value of $\Delta\mu$ and as a function of μ . --This was carried out for $\delta = 6^\circ$, dividing the total range of μ into approximately 250 equal intervals: $\mu_i = 0\ (\Delta\mu)\ \mu_{max}$, $\mu_{max} \sim 250 \cdot \Delta\mu$. As full use was made in (i) of the inherent symmetries of the problem, every frequency must be multiplied by an integer, which is 48 inside the reduced volume and correspondingly smaller for the faces, the various edges and vertices . Finally, to reduce the statistical fluctuations arising from the finite size of δ and $\Delta\mu$, a smoothing procedure was applied to the "curve" $N_i = N(\mu_i)$ (cf. 22.24).

(iii) Averages of Various Functions over $N(\mu)$

After the distribution function $N(\mu)$ has been computed, the averages of the following functions have to be found.

$$\left.\begin{aligned} A_\nu &= \frac{1}{\mu} \coth \frac{\alpha_\nu \mu}{2} \\ B_\nu &= \mu^{1/2} \left[\exp(\alpha_\nu \mu) - 1 \right]^{-1} \\ C_k &= \mu^k \end{aligned}\right\} \quad \left\{\begin{aligned} &\text{for 15 different values of } \alpha_\nu\text{:} \\ &\alpha_1, \alpha_2, \cdots, \alpha_{15} \\ &k = -2\ (\tfrac{1}{2})\ +2 \end{aligned}\right.$$

Since $\Delta\mu$ is finite, all integrals must be replaced by sums

$$\langle F \rangle = \frac{1}{N} \sum_i F(\mu_i + \tfrac{\Delta\mu}{2}) \cdot n_i$$

where

$$N = \sum_i n_i = (\pi/\delta)^3$$

and F stands for A_ν , B_ν , or C_k, respectively.

Both the A_ν's and the first C_k's are infinitely large for $\mu = 0$. The integrals still converge as $n(\mu)$ tends to zero, but our sums are becoming very

sensitive to and greatly affected by the statistical errors of n_i for small i. A special smoothing procedure had to be developed to overcome this difficulty (see 22.24).

22.22 Square Root Subroutines

Our code required the computation of very many square roots. It was largely due to the development of fast square root subroutines that the running time for this problem could be kept within quite reasonable limits. The total computing time, not counting in- and output, for one frequency distribution function is just 9 minutes. During this time 2792 matrices are computed, their eigenvalues determined, and the square roots of these eigenvalues are taken and distributed among 250 intervals according to size. The total number of square roots computed during these 9 minutes is about 50,000.

About 40,000 of them had to be evaluated for the diagonalization of the matrix T by the Jacobi method. For every two-dimensional rotation, $\tan 2\varphi$ is first obtained from which $\cos 2\varphi$, $\sin 2\varphi$, $\cos\varphi$ and $\sin\varphi$ must then be computed. This requires the computation of $\sqrt{1 + (\tan 2\varphi)^2}$ and $\sqrt{1 + \cos 2\varphi}$ where $|2\varphi| \leq \pi/4$. The radicand is thus known to be in a quite restricted interval in which a reasonably good linear approximation may be used to reduce the number of iterations to only two.*)

We shall show below how a best linear approximation for $\sqrt{x}$ can be found for $a \leq x \leq b$.

Another square root routine was written for part (iii) of our problem where $\sqrt{\mu}$ was needed for equidistant points $\mu = \Delta\mu \cdot m$, $m = 1,2,\ldots, m_{max}$. If m is not too small, $\sqrt{m}$ will be a reasonably good estimate for $\sqrt{m+1}$, but a much better one can be easily obtained as shown in (ii) below. As only little

---

*) The same technique can be used for the computation of square roots of normalized floating point numbers, or of any number after sensing and counting by how many places the radicand may be shifted to the left without overflow.

accuracy was required for the smallest values of μ a single iteration, using the formula shown below, was sufficient in our case.

(i) Linear Approximation for $\sqrt{x}$, $a \leq x \leq b$.

Most square root subroutines are based on the iteration formula

$$\frac{1}{2}\left(w_i + \frac{x}{w_i}\right) \Rightarrow w_{i+1}$$

It is well known that $w_i \longrightarrow \sqrt{x}$ for $i \longrightarrow \infty$ and, in particular, if

$$w_i = \sqrt{x}\,(1+\varepsilon), \quad \varepsilon \ll 1$$

then

$$w_{i+1} \approx \sqrt{x}\,(1+\tfrac{1}{2}\varepsilon^2)$$

In any case, w_2 (and hence w_3, w_4, ...) is always too big and the error depends only on the relative accuracy of w_i.

Let us assume that we know lower and upper bounds for x:

$$a \leq x \leq b$$

Then the best choice for a constant initial value obviously is $w_i = (ab)^{1/4}$
The error will have its maximum at the ends of the interval where

$$\frac{w_1(a)}{\sqrt{a}} = \left(b/a\right)^{1/4} \quad \text{and} \quad \frac{w_1(b)}{\sqrt{b}} = \left(a/b\right)^{1/4}$$

The next iteration yields for these points

$$\frac{w_2(a)}{\sqrt{a}} = \frac{w_2(b)}{\sqrt{b}} = \frac{1}{2}\left[\left(b/a\right)^{1/4} + \left(a/b\right)^{1/4}\right]$$

In order to get best relative accuracy over the whole interval we divide by the square root of this expression viz.

$$\theta = \left\{\frac{1}{2}\left[\left(b/a\right)^{1/4} + \left(a/b\right)^{1/4}\right]\right\}^{1/2}$$

We thus have

$$w_2 = \frac{1}{2\theta} \cdot \left(w_1 + \frac{x}{w_1} \right) = C_o + C_1 x$$

where

$$C_o = \frac{1}{2\theta} \cdot (ab)^{1/4} \; ; \; C_1 = \frac{1}{2\theta} (ab)^{-1/4},$$

The maximum error ratio is θ for $a \leq x \leq b$. It can easily be shown that this is the best linear approximation for $\sqrt{x}$ in the given interval. In our code this was used for b/a = 2, but the method pays off even better for wider intervals as the following table shows.

b/a	θ	b/a	θ
2	1.007498	10^4	2.247221
4	1.029884	10^5	2.986556
10	1.081809	10^6	3.978341
100	1.318807	10^7	5.303391
1000	1.703121	10^8	7.071421

(ii) Algorithm for a Table of $\sqrt{x}$

This algorithm is based on the approximation formula

$$\sqrt{\frac{1+u}{1-u}} \approx \frac{1+u/2}{1-u/2} = 1 + \frac{2u}{2-u}$$

which holds for $u \ll 1$ with an error of $\Delta \approx u^3/4$. Substituting $1/(2m+1)$ for u we obtain

$$\sqrt{m+1} \approx \sqrt{m}\left(1 + \frac{1}{2m+1/2}\right)$$

with a relative error of

$$\Delta \approx \frac{1}{32} \cdot \left(\frac{1}{m+1/2} \right)^3$$

for $m \gg 1$

Numerical example:

$$\sqrt{50} \approx \sqrt{49}\left(1 + \frac{1}{98.5}\right) = 7.0710660$$

exact value = 7.0710678

absolute error = .0000018

relative error = .00000026

A standard iteration would bring this relative error down to one half its square, which is less than 4.10^{-14} for our example.

22.23 Checking Procedures

Particular care has been taken to detect and trace errors due to false coding or machine trouble. We wish to outline some principles without going into too much detail.

(i) No attempt was made to minimize the number of locations used for temporary storage; on the contrary, as many different locations as feasible were used for the various variables so that a maximum of information about the immediate past was available at any given moment.

(ii) A subroutine was written and incorporated which would reconvert to decimal and punch out on cards the whole field of temporary storage ("Post Mortem Routine").

(iii) Transfer to this subroutine was made (a) manually during debugging after each "portion" of newly tested code (b) automatically if one of the mathematical checks, incorporated in the code, would fail.

The most important mathematical check consisted of squaring each frequency μ just after being computed and to test the vanishing of the determinant $|T - \mu^2 I|$. If the value was found to be bigger than 10^{-9}, transfer to the Post Mortem Routine was made. This happened just once, during production, and it was possible to trace the error to a multiplication, though some hundred operations had been carried out by the machine before the error was detected by the determinant check. Both factors together with the faulty result could be found, thus providing the maintenance engineer with accurate information.

In addition to these automatic checks several mathematical checks were carried out manually, to test the code and the accuracy of the various approximation procedures. For example, the third part (see 22.21-(iii)) was run for a "distribution" $n_i = 1$ for $0 \leq \mu_i \leq a$ and $n_i = 0$ beyond $\mu_i = a$. The values of $< F >$ thus obtained were compared with the integrals computed analytically.

22.24 Smoothing and Graphing

Due to the finite number of points for which the eigenfrequencies were computed the statistical fluctuations of the n_i were rather disturbing because they concealed the true character of the distribution function. Therefore, a subroutine was written which would replace each n_i by

$$\bar{n}_i = \frac{1}{4}(n_{i-1} + 2n_i + n_{i+1})$$

This "smoothing" could be repeated an arbitrary number of times, but after several applications, and before the "noise" had virtually disappeared, the characteristic and desired "true" discontinuities of the curve were lost. The easiest and most efficient way to find the best compromise was to use the graphing unit which had just been completed for our machine. The n_i, $\bar{n}_i$ $\bar{\bar{n}}_i$, ... were displayed on the screen of a cathode-ray tube by help of an auxiliary code and the optimum amount of smoothing could be determined by Dr. Squires himself by visual inspection.

A much more serious smoothing problem arose in part (iii) of the code. It can be shown from the power series development of the matrix T that the power series for $n(\mu)$, at $\mu = 0$, can contain only even powers of μ,

$$n(\mu) = C_0\mu^2 + C_2\mu^4 + \ldots$$

On the other hand, the A_ν and C_{-2} grow like μ^{-2} on approaching $\mu = 0$. Hence, the integrand theoretically tends to a constant, but numerically it does not because of the "noise" connected with the n_i. This noise is proportional to $\sqrt{n}$, hence to μ; when multiplied by μ^{-2}, for C_{-2}, it will increase like μ^{-1} for small values of μ.

Some of these difficulties could have been overcome by replacing the n_i by $n(\mu_i)$ computed from the power series. The amount of work -- coding and machine time -- to compute the coefficients C_{2k} of the power series is tolerable for k = 0 but grows rapidly for positive k. The second difficulty arises at the joining point of the analytic curve $n(\mu)$ with the original set of points. Let us examine it more closely.

We recall that the eigenfrequencies μ have been computed for the meshpoints only. Each meshpoint "represents" a certain volume, in general a cube. Each of the three eigenfrequencies of the matrix will be, for all points of the cube, pretty close to those for its center, i.e. the meshpoint, but not necessarily close enough to be in the same interval $(\mu_i \pm \frac{1}{2}\delta\mu)$ as the corresponding eigenfrequency for the meshpoint. This is the reason for the "noise" in our curve n_i.

Let us now look at our intergrals which are represented by sums (see 22.21.iii). The straight intergral

$$\int n(\mu)d\mu = \frac{1}{N}\sum_i n_i = 1$$

is not affected by the fluctuations, as the total count does not depend on the exact location where the eigenvalues are listed. If, however, $n(\mu)$ is multiplied by some function as e.g. $f(\mu) = \mu^2$, then

$$\int n(\mu)\mu^2 d\mu \approx \frac{1}{N}\sum_i n_i\mu_i^2$$

will hold only approximately. But even in this case, the effect of having a number n of counts erroneously shifted to the next interval will cause a relatively small error, viz.

$$n(\mu_i+\delta\mu)^2 - n\mu_i^2 \cong n \cdot 2\mu_i\delta\mu$$

Let us go back now to our idea of replacing the n_i by some analytic expression $n(\mu_i)$ up to, say, $\mu=\mu_k$ and let us assume again that a number n of counts should be in the interval $(\mu_k \pm \frac{1}{2}\delta\mu)$ but are, by error, in the next one. In this case the error will be

$$n(\mu_k + \delta\mu)^2 \approx n\mu_k^2$$

since n_k has been replaced by the fixed value $n(\mu_k)$ computed from the power series. This error is much bigger than the first one since $\delta\mu \ll \mu_k$.

Therefore, it was decided to join the analytic function $n(\mu_i)$ to the "statistical points" n_i not in one point but gradually by blending; the new values $\bar{n}_i$ were thus computed as

$$\bar{n}_i = g \cdot n(\mu_i) + (1-g)\, n_i$$

where the "weight function" $g(\mu)$ is defined by

$$g(\mu) = \begin{cases} 1 - \mu/\mu_k & \text{for } \mu \leq \mu_k \\ 0 & \text{for } \mu \geq \mu_k \end{cases}$$

and

$$n(\mu) = C_0\mu^2 + \bar{C}_2\mu^4 \quad ;$$

C_0 was computed directly by help of a modified code replacing each matrix element by the first term of its respective power series in α_1, α_2, α_3. $\bar{C}_2$, however, is a "mean value" computed from the original $n_i (0 \leq i \leq k)$ by the least square method using the same weight function $g(\mu)$ as above.

A more detailed analysis has shown that this whole procedure should reduce the error for the first part : $0 \leq \mu \leq \mu_k$ (actually about a quarter of the whole interval) to roughly the same amount as for the rest of the interval.

The development of this smoothing process, together with the special code for computing C_0, was rather time consuming, yet essential in order to obtain the final answer with sufficient accuracy.

22.30 NUMERICAL CALCULATIONS OF THE ANGULAR DISTRIBUTIONS FOR THE DEUTERON-PROTON AND SIMILAR REACTIONS.

Originator: William Tobocman
Analysist : Hans J. Maehly
Coders : Sonja Bargmann and Patricia Eberlein

When a light nucleus (e.g. a deuteron) hits a heavier one with sufficient energy to overcome the repellent Coulomb force, i.e. to enter into the range of nuclear interactions, one part of the light nucleus (e.g. a neutron) may be captured by the target nucleus, while the rest (the proton) will leave, accelerated by the repellent force, and at an angle θ with respect to the direction of the incoming particle. In the experiment, a virtually unidirectional and mono-energetic beam of light nuclei is aimed at a target of heavier ones and the density $\sigma(\theta)$ of outcoming residual particles (protons) is observed.

This "differential cross section" $\sigma(\theta)$ could be computed if the nature and laws of nuclear forces were known; as they are not, trial and error methods are used to confirm, improve or reject initial guesses for these forces. The numerical computation necessary to determine $\sigma(\theta)$, even according to a simple "guess" of the forces are, however, very tedious in most cases. Rather rough approximations have been used to reduce the work to a tolerable amount, and error estimates are so difficult that the conclusions drawn from such rough computations seem doubtful in some cases.

To overcome these difficulties, Dr. Tobocman made the following suggestions[1]:

(i) Not to start with the nuclear forces themselves, but with their effects (phase shifts) at a certain distance R from the center of the target nucleus, where R is approximately equal to its radius.

---

1) W. Tobocman and M. H. Kalos, Numerical Calculation of (d,p) Angular Distributions. Phys. Rev. 97 (1955), 1,p.132-136.

(ii) To prepare a most general and very flexible code which can be used, after its completion, for a wide range of initial data (describing the experiment) and of theoretical assumptions.

Work for this code began in summer 1955 with an extensive study of the various mathematical problems. Programming and coding was done through the year 1956. The code is now in the debugging and testing phase. It is expected to be in full operation in summer 1957.

About 90% of this work was supported by Contract No. Nonr-1358-(03), the rest by the Army Contract for which this volume constitutes the Final Report. We shall therefore prepare a Technical Report to the Office of Naval Research at the completion of the problem.

22.40 DISTRIBUTION OF EIGENVALUES OF BORDERED MATRICES WITH INFINITE DIMENSIONS

Originator: Eugene P. Wigner

The problem to determine the statistical properties of the characteristic values of infinite bordered matrices arose from the consideration of the properties of the wave functions of quantum-mechanical systems which are assumed to be so complex that statistical considerations can be applied to them.[1] The mathematical problem together with an integral equation and asymptotic formulae for the distribution of the eigenvalues and with a recurrence formula for their moments are given in a paper by WIGNER[2], to which we would like to refer the reader interested in the details of the following paragraphs.

Our task was to find a method and to write and run a code for the numerical computation of the distribution of the eigenvalues for various values of the parameter q (cf eqs. (1) and (4) below). The results so far obtained are not fully satisfactory, but other urgent work prevented us from improving them.

22.41 Two Possible Approaches

(i) Method of the Integral Equation.

WIGNER (loc.cit.) derived the following non-linear integral equation for the distribution $\rho(x)$:

$$\rho(x) = q\left\{R(x)\cdot\int_{-\infty}^{+\infty}\frac{\rho(z)-\rho(x)}{x^2-z^2}\,dz + \rho(x)\cdot\int_{-\infty}^{+\infty}\frac{R(z)-R(x)}{x^2-z^2}\,dz\right\} \qquad (1)$$

where

$$R(x) = \int_{x-1}^{x+1}\rho(\xi)\,d\xi \qquad (2)$$

---

1) J. A. Lane, R. G. Thomas and E. P. Wigner, Giant Resonance Interpretation of the Nucleon-Nucleus Interaction, Phys.Rev. 98 (1955), pp.693-701.

2) E. P. Wigner, "Characteristic Vectors of Bordered Matrices with Infinite Dimensions," Annals of Mathematics 62 (1955), p. 548.

Not much, if anything, is known about this kind of integral equation. But it was hoped that the following iterative procedure might yield an adequate approximation to the distribution function.

We start with a trial function $\rho_0(x)$ which may be constructed by interpolating very roughly between the asymptotic solutions for very small and very large values of q, respectively, these asymptotic solutions being known from Wigner's paper. We then compute $R_0(x)$ from (2) and $\rho_1(x)$ from (1), substituting ρ_0 and R_0 for ρ and R on the right-hand side, and so on. The integrals are of course replaced by sums (e.g. using Simpson's rule) and the limits in (1) by some finite values $\pm X$. From the known asymptotic behavior of $\rho(x)$, X can easily be determined so that $\rho(x)$ and R(x) are numerically negligeable for $x > X$.

The results of this approach were unsatisfactory. It was believed that this was due to some basic difficulty in the procedure. Hence another method, described below, was developped; but it should be said that meanwhile errors have been found in the first code so that, after all, the iteration method described above may work if properly coded.

(ii) Method of the Moments

The second method does not use the integral equations (1) and (2) but a recurrence formula for the moments of the distribution $\rho(x)$ which is also found in Wigner's paper (loc. cit.):

$$\mu_{2\nu} = \int_{-\infty}^{+\infty} \rho(x) x^{2\nu} dx \tag{3}$$

$$\mu_{2\nu} = \delta_{0\nu} + 2q \cdot \sum_{\varkappa=0}^{\nu-1} \sum_{\lambda=0}^{\varkappa} \binom{2\varkappa+1}{2\lambda} \mu_{2\nu}\mu_{2\nu-2\varkappa-2} \tag{4}$$

All odd moments vanish, as $\rho(x)$ is an even function. It is easy to compute $\mu_{2\nu}$ for, say, $\nu = 0....50$ from (4), particularly since all contributions are positive. However, the solution of the moment problem, i.e. computing $\rho(x)$ from the $\mu_{2\nu}$, is, numerically, a very unstable procedure. We finally succeeded in getting fair results for several values of q, but the method would fail for both very large and very small values

of this parameter. Fortunately the known asymptotic solutions just fill this gap.

22.42 Numerical Solution of the Moment Problem

Our general approach is to try to fit to the unknown function a step function having as many moments as possible correct. We use step functions with a fixed length of step (denoted by h) and proceed in the following fashion: for our first approximation we take a step function with one step such that its integral is equal to the integral of the unknown function. Next we take a function having three steps such that its integral and its second moment are correct. For our n-th approximation we take a symmetric step function having 2n+1 steps such that its first 2n moments and its integral are correct. Thus, at each step, we are solving n linear equations in n unknowns. If this sequence of functions converged, the limit would be a function with the correct moments, and therefore the unknown function. However, in practice it does not converge, and therefore we will stop the process at some point where we believe to have obtained the best approximation to the function. To determine this point, we use the following rather heuristic argument.

With our n-th step function we are matching the first 2n moments of the unknown function with a function of extent approximately from -2nh to +2nh, constant over intervals of length h. The extent of our step functions, therefore, goes up as 2n. With these functions we try to approximate the unknown in that portion of its domain which gives a significant contribution to the first 2n moments. This region does not in general go up linearly with n. If this region of significance is much smaller than the extent of the step function, it is clear that in effect we have not enough parameters in the region of significance to graph the function from 2n moments. This is why the infinite sequence of step functions diverges, for we have good reason to believe that the region of significance of our unknown function increases at a less than linear rate.

Hence our approach is to try and find a step function with 2n+1 steps such that its extent is approximately the same as the region of significance

of the unknown function for its first 2n moments. At the same time we want to make n as large as is feasible and h, the length of the steps, small enough to graph the function accurately.

We start by choosing a particular h, and proceed to apply the method of approximation described above. But instead of using simple step functions s_n defined by

$$\left.\begin{aligned} s_n = 1 \text{ if } \left(-n-\tfrac{1}{2}\right) \leq \tfrac{x}{h} \leq \left(-n+\tfrac{1}{2}\right) \text{ or } \left(n-\tfrac{1}{2}\right) \leq \tfrac{x}{n} \leq \left(n+\tfrac{1}{2}\right) \\ S_n = 0 \text{ otherwise,} \end{aligned}\right\}$$

we take linear combinations of these with the property that the first 2n-1 moments are zero, while the (2n)th moment is not. This puts the system of equations-that we solve at each step-into triangular form. As long as the region of significance of the unknown function is larger than the extent of the approximation, the correction to the (2n)th moment will be positive, that is the (2n)th moment of the approximation will be smaller than that of the unknown function. As soon as this is no longer the case, we stop and take the approximation at that step as our best fit for our particular choice of the step length h.

For each value of q we have used several values of h, trying to obtain as high a density of values as possible in at least the intermediate range of x. The results were especially good in the range of q between .1 and 1.0 Since the value of h could not be taken too small (there seemed to be no reasonable behavior at all for very small h) we could not obtain points close to zero. However, in the range mentioned the points obtained seemed sufficient to give a good picture of the function.

For very small $q < .1$, the behavior around zero is much more active, the function acquires a steeper and steeper peak which is impossible for us to graph accurately with this method. For $q > 1$, our technique works poorly and we got fewer and less reliable, as well as less likely points.

22.50 MOLECULAR INTEGRALS

Originator: R. C. Sahni
Analyst and Coder: J. W. Cooley

Many numerical computations have been carried out to determine the energy levels of atoms from the SCHROEDINGER equation and the relevant atomic constants, and the comparison of these accurately computed theoretical values with the observed atomic spectra has greatly advanced our knowledge of the laws governing the electron shell; through the analysis of the hyperfine structure, it has been an important tool to determine the magnetic moments of many nuclei.

The analysis of molecular spectra, however, has been hampered by the fact that the numerical computations necessary to obtain the required accuracy are beyond the limits of desk computing in all but the very simplest cases (such as H_2). These computations constitute, therefore, a promising field in which to use electronic digital computers. They are, however, by no means straightforward. The proper selection of atomic orbits requires a great deal of experience and thorough knowledge in this special field; careful and tedious mathematical analysis is necessary to keep the numerical work down to what is tolerable even for a fast electronic computer and to guarantee sufficient accuracy by avoiding numerically instable procedures.

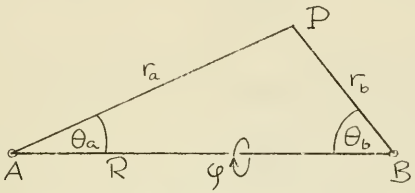
R. C. Sahni and J. W. Cooley have attacked this problem, using our computer for the numerical work. While computer operation was supported by Contract Da-36-034-ORD-1646, for which this final report has been prepared, all analytic and coding work has been supported by Contract NAw 6475 (between the National Advisory Committee on Aeronautics and New York University) under the terms of which technical reports will be prepared. Therefore, we shall not report on the results of this work, but merely mention some aspects of the computations which may be of more general interest.

22.51 Decomposition of Overlap Integrals

All molecular integrals have integrands which depend on the distances r_a and r_b from two nuclei A and B. A typical example is the overlap integral:

$$S(\psi_a, \psi_b, R) = \int \psi_1(r_a, \theta_a, \varphi)\, \psi_2(r_b, \theta_b, \varphi)\, dV$$

The meaning of the coodinates can best be shown by a sketch



and in the simplest case, the functions ψ_a and ψ_b may be a pair of SLATER orbitals such as:

$$\psi_{1s} = (k^3/\pi)^{1/2} e^{-kr}$$

$$\psi_{2s} = (k^5/3\pi)^{1/2} r e^{-kr}$$

$$\psi_{2px} = (k^5/\pi)^{1/2} r e^{-kr} \sin\theta \cos\varphi$$

etc.

The k's of ψ_a and ψ_b are sometimes the same, but more often they are different.

It would be possible, of course, to carry out the integrations "by brute force", i.e. by computing the integrand explicitely in three-dimensional space for a sufficient number of points to achieve the desired accuracy. This method would be neither attractive nor economical. Fortunately, the angle φ is common to the A and the B systems and the integration with respect to φ can be easily carried out. We are thus left with a two-dimensional integral. For the SLATER orbitals, this can be further reduced to linear combinations of one-dimensional integrals [1] A_n and B_n, as for example:

---

1) cf. R. S. Mullikan, C. A. Rieke, D. Orloff and H. Orloff, Journal of Chem. Phys., 17, 1248-1267,(1949) where numerous further references are given.

$$S(\psi_{a1s}, \psi_{b2s}) = \frac{p^4}{8\sqrt{3}}(1+t)^{3/2}(1-t)^{5/2}\left[A_3B_0 - A_2B_1 - A_1B_2 + A_0B_3\right]$$

where
$$A_n = \int_1^\infty u^n e^{-pu}\,du = A_n(p)$$
$$B_n = \int_{-1}^{+1} v^n e^{-ptv}\,dv = B_n(pt)$$

and
$$p = \tfrac{1}{2}(k_a + k_b)R \quad ; \quad t = \frac{k_a - k_b}{k_a + k_b}$$

This reduction is achieved by introducing spheroidal coordinates

$$u = \frac{r_a + r_b}{R} \quad ; \quad v = \frac{r_a - r_b}{R}$$

22.52 Computation of the Auxiliary Integrals A_n and B_n

The "master formulae" expressing the SLATER atomic orbit overlap integrals in terms of A_n's and B_n's are so simple that they can be handled on desk computers. It is worth while, therefore, to produce tables of the A_n and B_n integrals. Preliminary tables for n = 0 (1) 5 and a wide range of the argument have been computed and more extensive tables are planned for publication. The following procedures have been employed in order to get at least eight digit accuracy:

(i) The $A_n(p)$ can be easily computed by partial integration, which yields the recurrence formula

$$A_n(p) = \frac{1}{p}\left[e^{-p} + nA_{n-1}(p)\right]$$

where
$$A_0(p) = \frac{1}{p}e^{-p}$$

This procedure is numerically stable since all terms are positive.

(ii) By partial integration of the B_n integral one obtains

$$B_n(q) = \frac{1}{q}\left[(-1)^n e^q - e^{-q} + nB_{n-1}(q)\right]$$

where
$$B_0(q) = \frac{1}{q}\left(e^q - e^{-q}\right) = \frac{2}{q}\sinh q$$

The recurrence formula contains both positive and negative terms and is unstable except for large values of the argument q.

(iii) For small values of q, $B_n(q)$ may be computed from the power series

$$B_{2m}(q) = 2\sum_{j=0}^{\infty} \frac{q^{2j}}{(2m+2j+1)\cdot(2j)!}$$

$$B_{2m-1}(q) = -2\sum_{j=0}^{\infty} \frac{q^{2j+1}}{(2m+2j+1)\cdot(2j+1)!}$$

These series are numerically stable but require much more work than the recurrence formulae, especially if q is not small. Experiments have shown that the recurrence formula yields the same results, within the desired eight digits accuracy, if $q>1$ and $n\leq 5$, and,therefore, it will be used for the final tables within this range.

22.53 Numerical Integration

We have seen that the overlap integrals can be expressed by the A_n's and B_n's, which are relatively easy to evaluate. Two other types of molecular integrals, viz. the Coulomb and Hybrid integrals, can be computed with the help of the auxiliary function[1)]

$$Z_{n\ell}(k,\tau) = \int_0^{\infty} \zeta_n(t,\tau) e^{-kt} t^{\ell+\frac{1}{2}} dt$$

where

$$\zeta_n(t,\tau) = \begin{cases} I_{n+\frac{1}{2}}(t)\cdot K_{n+\frac{1}{2}}(\tau) & \text{if } t\leq\tau \\ I_{n+\frac{1}{2}}(\tau)\cdot K_{n+\frac{1}{2}}(t) & \text{if } t\geq\tau \end{cases}$$

---

1) Barnett, M. P. and Coulson, C. A. Phil. Trans. Roy. Soc. London 243 (1951) p. 221

The computation of the Bessel functions $I_{n+1/2}$ and $K_{n+1/2}$ presents no difficulty. The recurrence formula

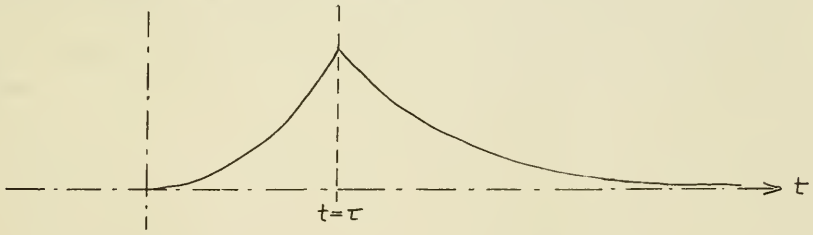
$$K_{n+3/2}(z) = K_{n-1/2}(z) + \frac{2n+1}{z} \cdot K_{n+1/2}(z)$$

is convenient, and it is stable for all positive values of z. The corresponding recurrence formula for $I_{n+1/2}(z)$ is not stable for small z. Hence the power series

$$I_{n+1/2} = \frac{(2z)^{n+1/2}}{\pi^{1/2}} \cdot \sum_{j=0}^{\infty} \frac{(n+j)! \cdot z^{2j}}{j!\,(2n+2j+1)!}$$

was used for $z < 10$ and $n > 1$.

The integral $Z_{n\ell}$ must then be evaluated by numerical integration. The integrand is positive throughout, except for t=o, where it vanishes like $z^{n+1/2}$; it will further decrease exponentially for $x \longrightarrow \infty$. For $t=\tau$ the integrand is continuous, but the first derivative is not. Thus the integrand has, roughly, the following shape



A number of integration procedures, such as Weddle's rule, Gaussion quadrature and Euler's formula, were considered, and the last one was finally chosen, as it seemed to be not only the simplest but also the fastest procedure for a given accuracy. As is well known, Euler's formula can be described as integrating, for each interval, a third degree polynomial whose values and first derivatives at both ends agree with those of the actual curve, while the trapezoidal does not adjust the derivative at all. Yet, the only additional terms for Euler's formula are the first derivatives at the end points and, of course, at such points where the first derivative is discontinuous. This was easy to code and the results were fully satisfactory, as could be seen by recomputing the most critical cases with narrower intervals.

22.54 Coding Procedure

As usual, we first studied the question of whether to use direct machine language or the FLINT interpretive routine. Since it was evident that serious scaling problems would arise and that card punching would take up an appreciable fraction of the total machine time, it was decided to use FLINT, but to speed up the output by changing the output routine.

The flexibility of FLINT permitted a good deal of experimenting in developing methods, and debugging was fast and easy. The numerical integration, however, required quite a lot of machine time. The two innermost loops which, of course, take up most of the running time, were then rewritten in "direct floating point", i.e. retaining the floating point arithmetics but saving the time for interpretation and some unnessary normalization by recoding these loops in machine language. This proved to cut machine time down to about one fourth without requiring too much coding time.

The normal FLINT output routine puts only two floating point numbers on each card and punches only one card at a time. The new routine increased these figures to three and eight, respectively. This not only reduced the output time by appr. 60%, but also produced a format far more suitable for our tables.

22.60 NUMERICAL EVALUATION OF SOME DOUBLE AND TRIPLE INTEGRALS ARISING FROM MESON THEORY

Originator: Hiroshi Suura

Analysis and Coding: Hans J. Maehly and Hiroshi Suura

Recent experiments on high-energy electron scattering by hydrogen show that the proton has an extended structure, as expected from meson theory, and that the measured cross-section can be fitted approximately by assuming a certain distribution of the proton charge, and the magnetic moment for this distribution. Dr. Suura has derived rigorous expressions for these distributions from meson theory. Each distribution is expressed as the sum of a double and a triple integral.

As an example we show one of the triple integrals

$$I(q) = \frac{1}{72\pi^3}\frac{M}{\mu}\iiint\left\{\frac{\sigma(z)}{(\sqrt{1+x^2}+\sqrt{1+z^2})(\sqrt{1+y^2}+\sqrt{1+z^2})}\left[\frac{1}{\sqrt{1+z^2}}+\frac{1}{\sqrt{1+x^2}+\sqrt{1+y^2}}\right]\times\right.$$

$$\left.\times\frac{x}{\sqrt{1+x^2}}\cdot\frac{y}{\sqrt{1+y^2}}\cdot\frac{2(x^2y^2+q^2x^2+q^2y^2)-x^4-y^4-q^4}{q^3}\right\}dxdydz$$

where

$$\sigma(z) = \frac{12\pi z^4}{\omega^2(A-\omega B)^2+z^6}$$

and

$$\omega = \sqrt{(M/\mu)^2+z^2}+\sqrt{1+z^2}-M/\mu$$

and the region of integration is defined by the inequalities:

$$0 \le x \le b;\quad \left\{\begin{array}{c} q-x \le y \le q+x \\ x \le y \le b \end{array}\right\};\quad 0 \le z \le b.$$

It can easily be seen from these inequalities that $0 \le q \le 2b$.

The integrals were approximated by sums of the integrand at points:

$$\left\{\begin{array}{l} x = (i+\frac{1}{2})\cdot\delta \\ y = (j+\frac{1}{2})\cdot\delta \\ z = (k+\frac{1}{2})\cdot\delta \end{array}\right\}\quad \delta = \frac{b}{n}$$

i, j, k being integer and subject to inequalities analogous to those for x, y, z.

For convenience, the same interval δ was used for q, i.e. the integrals were computed for

$$q = (\ell + \tfrac{1}{2})\cdot\delta \,, \quad \ell \text{ integer}, \quad 0 \leq \ell \leq 2n-1 \quad .$$

The integer n and hence the interval δ were not fixed constants but part of the input; the code was run for various values of n and it was found that the accuracy obtained with $\delta = 0.2$ was fully satisfactory. The computations were then carried out for b = 5.4 and b = 6.0.

At a first glance, the number of points for which the integrand must be computed seems to be of the order of n^4, requiring about one hour computing time. However, the integral given above as a typical example can be written as

$$I(q) = \frac{1}{72\pi^3}\,\frac{M}{\mu}\iint \frac{2(x^2y^2+q^2x^2+q^2y^2)-x^4-y^4-q^4}{q^3}\,J(x,y)\,dx\,dy$$

where

$$J(x,y) = \int \frac{\sigma(z)}{(a+c)(b+c)}\cdot\left(\frac{1}{c}+\frac{1}{a+b}\right)\frac{xy}{ab}\,dz$$

and

$$a=\sqrt{1+x^2}\,, \quad b=\sqrt{1+y^2}\,; \quad c=\sqrt{1+z^2}$$

This decomposition brings the number of integration points down to the order of n^3 and the computing time to 3 minutes, excluding input and output. This is so short that we decided to use the Floating Point Interpretive Routine (FLINT) with an estimated running time of 30 minutes.

The times for coding, debugging and production are fairly typical for FLINT. Coding, from a finished flow chart, took one afternoon. Debugging was done for n=3, using the FLINT-TRACER. A few trivial errors were readily found and corrected, all this on the evening of the same day; machine time: approximately 20 minutes. Production took place the next evening, requiring another 40 minutes of machine time. Dr. Suura watched the coding for the first two integrals and then easily coded the remaining two without assistance from our staff.

23. PROBLEMS IN VARIOUS FIELDS

23.10 TRAFFIC SIMULATION WITH A DIGITAL COMPUTER

Originator and Coder: S. Y. Wong

Every one of us has had the experience that the capacity of a traffic system is determined by its weakest point. Millions may be spent on the construction of a third lane, while the real cause of traffic jams lies in a quite restricted area, say, one traffic circle. It is possible, of course, to find and remedy these bottlenecks by the method of trial and error. But it would be much more economical to replace at least part of this experimentation by automatic computing. The two main difficulties hampering a computational approach are

(i) to determine the "input parameters", i.e. the expected flow of traffic at all points and under various conditions (weekdays, holidays, etc.)

(ii) to develop a suitable "model" of the system under study in terms of discrete variables including the mechanical properties of the cars and the psychological behavior and reactions of the drivers.

It has been the purpose of the present study to demonstrate that a fast automatic digital computer is well suited to carry out the actual computation after the above difficulties have been overcome. A very simple example for which the difficulties (i) and (ii) are reduced to a minimum was carried out numerically.

23.11 Description of the Simulation System

The read Section under study was a six lane one-way street, divided into two three-lane sections. The dividing island had one gap which allowed for changing from the right-hand to the left-hand side. Each lane was, for computational purposes, divided into unit blocks (UB) of "unit car length" (UCL) = 18'4", while the time step was chosen to be one second. This yielded a speed unit of 18'4"/sec = 12.5mph. For each time step the cars were moved by a discrete number of UCL's, beginning with the cars closest to the "exit" in order to avoid double occupancy of UB's. The movement was further controlled by some traffic regulations (such as $v \leq 50$mph) and a

certain probability distribution for the generation of random numbers governing the input of cars, their respective speed and intention to switch lanes.

23.12 Conclusion

The results show that the flow of traffic resembles the observed behavior to a fair degree, in spite of all simplifications, and that the distribution of transit times does not depend too much on the specific set of random numbers even for rather short runs (several minutes). Much longer runs would be required, however, for a realistic, i.e. more complicated and extended road system. It is obvious, therefore, that faster machines with larger high-speed memories would be needed to solve actual problems. Such machines are, or will soon be available. It will be well worth while, therefore, to further pursue this application of fast automatic computing and to focus some research on the problems (i) and (ii) mentioned in 23.10 above.

23.20 A MIXING PROBLEM

Originator and Analyst: H. H. Goldstine
Coder: Mrs. M. Lamb

This problem is concerned with two compressible fluids, one on top of the other in a two dimensional container with rigid walls. The only exterior force acting is that of gravity. At the beginning of the problem, each fluid will be in hydrostatic equilibrium except for a small sinusoidal perturbation of the interface.

If the upper fluid is lighter than the lower fluid at the same pressure the problem is almost trivial. The aim of our computation was to determine the sequence of events in the reversed case. We shall expect, of course, that the lighter fluid will "break through" at a relatively small front and then spread out on the surface above the upper fluid.

The problem was formulated in Lagrangian form. Accordingly, at each meshpoint five quantities were stored, namely two space coordinates, two velocities, and the energy. It was found necessary to use about 600 meshpoints, hence to store around 3000 quantities. As the new drum was not yet completed at that time it was necessary to store two numbers (of 20 binary digits) in each word.

In order to avoid the formation of shocks, which are not relevant to the main study, the method of von Neumann and Richtmyer[1] was adapted to our case.

It was planned from the beginning to proceed step by step from a very simple test case (with the lighter fluid on top) to more realistic and more complicated problems. However, great difficulties were encountered in formulating this problem and the work was greatly hampered by the limited size and poor reliability of the old magnetic drum. The problem was finally abandoned when both Dr. Goldstine and Mrs. Lamb left this project.

---

1) A method for the Numerical Calculation of Hydrodynamic Shocks, J. Appl. Phys. 21, 232-237 (1950)

23.30 NUMERICAL INTEGRATION OF THE NAVIER-STOKES EQUATIONS FOR COMPRESSIBLE FLUIDS

Originator, Analyst and Coder: S. H. Lam

It is attempted to obtain a numerical solution to the complete system of partial differential equations governing the steady flow field of a compressible, viscous and heat conducting gas in two dimensions. The physical problem selected for study is the motion of a thin flat plate moving with hypersonic speed at zero angle of attack. The major interest of this problem is the behavior of the solution about and ahead of the leading edge, where no analytical solution, exact or approximate, is known. The asymptotic behavior of the solution at downstream infinity, however, is known from the approximate boundary layer theory.

The plate shall be insulated and of zero thickness, and the "non-slip" condition on the plate is assumed to be valid. The plate is semi-infinite.

23.31 Nature of the Solution

Since this is a physical problem, the general character of the solution may be inferred from the experimental data. We thus expect that the solution fields will be essentially undisturbed far ahead and far away from the plate. A curved detached shock wave will stand at some distance in front of the leading edge of the plate. A boundary layer surrounds the immediate vicinity of the flat plate, on the surface of which the gas velocity and the normal temperature gradient are zero. At distances far downstream of the leading edge the flow field will behave according to the known boundary layer theory. Also the flow field in front of the curved shock wave will be practically undisturbed. The information to be obtained from the numerical solution concerns the shape of the shock wave, the detachment distance of the shock wave from the leading edge, the flow field about the leading edge, and the pressure distribution on the plate surface.

23.32 Equations Governing the Flow Field

Let the plate be situated on the positive X axis, with the leading edge at the origin. Then

$$\frac{\partial u}{\partial t} = -\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) - \frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial v}{\partial t} = -\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) - \frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

$$\frac{\partial \rho}{\partial t} = -\left(\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y}\right)$$

$$C_p\frac{\partial T}{\partial t} = -C_p\left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}\right) + \frac{1}{\rho}\left(\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y}\right)$$
$$+\left\{\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)\right\}$$
$$+\nu\left\{2\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right] + \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2 - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)^2\right\}$$

where

	u	velocity in +x direction
	v	" " +y "
	t	time
	T	temperature
	p	pressure
	ρ	density,
	ν	kinematic viscosity
	k	conductivity
	C_p	heat capacity

and

$C_p = C_p(T)$; $k = k(T)$; $\nu = \nu(T)$; $\rho = \rho(p.T)$

are known relations.

In essence we have, therefore, a set of four non-linear simultaneous partial differential equations for the four dependent variables, u, v, ρ , T. The character of the equations is elliptic in space and parabolic in time, and we must impose appropriate boundary conditions on a closed curve surrounding the region of interest. The above set of equations contains three independent variables, x, y, and the time, t. Although we are only interested in the steady behavior of the flow field, we have been using the unsteady equations to facilitate a scheme of successive approximation to be described later.

23.33 Main Difficulties

As was mentioned above, we expect a curved shock wave in the solutions. A shock wave occupies a narrow region in which the gradients of all dependent variables are very large. If the mesh of the flow field were made fine enough to permit an adequate representation of this narrow region, then the number of mesh points required would be prohibitively large. However, the structure of the shock is of no interest in this problem. It is only required that the magnitudes of the dependent variables before and after the shock can be related correctly as a function of the orientation of the shock. Actually, the exact relations of the dependent variables before and after a shock as a function of the shock orientation are known in analytic form. However, since the shape and position of the shock are not known to us, -- indeed, they are the objective of this investigation -- we cannot take advantage of these relations.

Our first problem, now, is to find a difference scheme which represents the derivatives in the equations such that the resulting system of difference equations is capable of yielding correct relations of the dependent variables before and after the shock for any shock orientation with a very coarse mesh, perhaps with only two or three mesh points. Furthermore, the difference equations must be stable in flow regions away from the shock.

The second difficulty is the correct choice of the initial and boundary conditions. If the mesh size selected is fine enough to be stable, yet coarse enough so that, with a reasonable number of mesh points, the top, front and downstream sides of the rectangle can be effectively considered to be $y = +\infty$, $x = -\infty$, and $x = +\infty$, respectively, with the origin at the leading edge, then the boundary conditions in the steady state would be essentially those described

in 23.31. However, since we choose to approach the steady state solution from an initial condition through a transient calculation, the appropriate boundary conditions to be imposed during transient are not known. This problem is still being studied analytically.

23.34 Simplified Problem

In order to gain experience and insight concerning our system of differential equations, a much simpler problem was proposed and coded as a trial problem. The dependent variable T, was assumed to be constant throughout the flow field at all times, thus reducing the number of dependent variables to three. Physically this corresponds to isothermal flow, or a gas of infinite heat conductivity. This eliminates the possibility of a shock wave, therefore the problem is much simpler.

The flow field is represented by a 32x32 grid, with the flat plate situated at the bottom of the rectangle. The leading edge is situated at the middle, and the flow is coming from the left.

The computation was started with the fictitious initial state of a uniform flow defined by $u = 1/4$, $v = 0$ and $p = 1/4$ for the entire region.

Using the time-dependent equations, one can calculate the flow field at $t = n+1$, knowing the flow field at $t = n$. For the initial condition of uniform flow the gas velocity on the plate is non-zero. As the computation begins, the boundary condition of non-slip, i.e. $u = 0$ on the plate surface, is gradually imposed. We used $u_{n+1} = u_n - 1/32$, with $u_o = 1/4$ and $u_n = 0$ for $n \geq 8$. The values of $u = 1/4$, $p = 1/4$, $v = 0$ are held rigidly constant with time on the upstream and top sides of the rectangle. On the portion of the bottom side of the rectangle in front of the leading edge, we require $v = 0$ and $\frac{\partial u}{\partial y} = \frac{\partial p}{\partial y} = 0$. The purpose of the trial problem is to study the stability of our difference equations and to obtain information about the appropriate boundary condition for the downstream side of the rectangle.

If n is the time index and k and m are the x and y space indices, respectively, then the difference schemes to represent the partial derivatives used are :

$$\frac{\partial V}{\partial t} = \frac{V_K^{n+1} - V_K^n}{\Delta t}$$

$$\frac{\partial V}{\partial x} = \frac{V_{K+1}^n - V_{K-1}^n}{2\Delta x} \qquad \frac{\partial V}{\partial y} = \frac{V_{m+1}^n - V_{m-1}^n}{2\Delta y}$$

$$\frac{\partial^2 V}{\partial x^2} = \frac{V_{K+1}^n - 2V_K^{n+1} + V_{K-1}^n}{(\Delta x)^2} \qquad \frac{\partial^2 V}{\partial y^2} = \frac{V_{m+1}^n - 2V_m^{n+1} + V_{m-1}^n}{(\Delta y)^2}$$

where V stands for any dependent variable. We always chose $\Delta x = \Delta y$, but $\Delta x/\Delta t$ was left free for adjustment as required for stability.

There is still very little known about the boundary condition to be applied on the downstream side, and before we know the analytical asymptotic solution during the transient, any inaccuracies in the boundary condition will generate errors which shall propagate upstream. In the trial runs $\partial^2 V/\partial x^2 = 0$ was tentatively used as a first approximation.

The result of the trial runs for $\frac{\Delta x}{\Delta t} = 2$ showed signs of instability in the difference scheme used. After about six or eight time cycles, it was found that scattered negative values of u, v, and p began to show up in the flow field, probably indicating numerical "overflow" in the machine, i.e. physically unreasonably large values. Up to this time cycle the errors from the approximate downstream boundary condition propagate about three mesh points upstream.

23.35 Plans for Further Investigations

It is planned to continue the search for a stable difference scheme for our system of partial differential equations. We hope to succeed in eliminating the source of trouble at the downstream boundary by finding and using the analytical asymptotic solution for $x \longrightarrow +\infty$. The present knowledge of the appropriate representation of partial derivatives in this non-linear equation system is very scanty and we have been forced to use trial and error methods. If a stable difference scheme can be found, the computation of solutions to this problem should be fairly straightforward. The experience and insight gained in solving this problem should help us to attack successfully a variety of compressible viscous flow problems in the near future.

23.40 AUTOMATIC NETWORK ANALYSIS

Originator: S. Y. Wong
Analyst and Coder: M. Kochen

The analysis of complicated electrical networks, for direct or for alternating current, has long been considered a stronghold of analogue computers since the construction of such network analysers is pretty much straightfoward. The application of digital computers, however, offers the great advantage that changes in the parameters or even in the connections can be made much faster and easier, once a general code has been prepared. It was the aim of the authors to investigate the practicality and limitations of digital computing in this field. The method is explained in a Technical Report.*) We therefore limit ourselves to a necessary correction.

As can be seen from page 9, the "power dissipation" P is a quadratic function of the voltages w_i. The Taylor series on page 11 terminates, therefore, with the quadratic term and so far no term has been neglected.

The statement on page 12 "Because the Taylor series was truncated after two terms, the above expression is only a first approximation to ΔV " is, therefore, wrong; actually ΔV was deliberately chosen to be parallel to grad P.

*) M. Kochen and S. Y. Wong, Technical Report No. 55-02, August 1955.

23.50 A TABLE FOR CUMULATIVE BINOMIAL PROBABILITIES

Originator: J. W. Tukey
Coder : H. F. Trotter

The probability that in n independent trials of an event having probability p of "success" the number of successes will not exceed k is the cumulative binomial probability

$$P(p, n, k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

This function is not well suited to ordinary interpolation procedures and since it has three arguments extensive tables are necessary. The object of our computation was to produce a short table in a modified form such that simple interpolation will give results of reasonable accuracy.

Define K(p, n, k), the "equivalent normal deviate", by the equation

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{K} e^{-x^2/2} dx = P(p,n,k)$$

Then

$$N(p, n, k) = 2 \left\{ \sqrt{(k+1)(1-p)} - \sqrt{(n-k) \cdot p} \right\}$$

is a fair approximation for K(p, n, k) and is used as an auxiliary variable for the table listing values of K-N as a function of p, E = np, and N, with the respective ranges:

$$\begin{aligned} p &= 0\ (0.1)\ 0.5 \\ N &= -4\ (0.5)\ +4 \\ 24/E &= 1\ (1)\ 24 \end{aligned}$$

As a result of our transformation, this rather coarse mesh will permit fairly accurate interpolation for most values of p, n, k, with the exception of a small region where a finer mesh would be desirable.

23.60 Experiments in the Use of FLINT

During the months of October, November, and December in both 1955 and 1956 experiments were carried out to test the effectiveness of the FLINT language as a tool for the inexperienced coder. Graduate and undergraduate engineers from Princeton University, without prior experience in digital computing, were encouraged to code two standard problems in order to find out what difficulties they encountered. They coded the solution of n simultaneous linear algebraic equations and the intergration of an n^{th} order ordinary differential equation (Milne, initial condition). These routines are now available as service routines.

The experiments showed remarkably fewer coding errors than have been observed in machine language codes for other machines -- apparently attributable to a lack of exceptions in the FLINT language. The errors that did occur suggest desirable changes for any future external language that may be devised.

23.70 HISTORICAL EPHEMERIS FOR THE YEAR -600 to 0

Originator: Otto Neugebauer and A. Sachs
Analysist and Coder: Bryant Tuckerman

Among the tools of historical research is the possibility of establishing chronology by means of ancient astronomical records. This technique applies whenever it can be shown, from modern astronomical theory, that an observation or configuration recorded in an ancient text could only have occured on a particular date. This dates the observation, and hence generally the text, the calligraphy, and any contemporary historical and social records in the text.

Some ancient observations have even been precise enough to furnish improvements in the values of some astronomical quantities - the empirical secular accelerations - which cannot be predicted from theory.

Heretofore methods have not been available for directly choosing possible dates from the given observations. Instead it has been necessary to choose provisional dates, compute the positions of the desired bodies from existing theories, compare the results with the recorded observations, and use the results of the comparison to estimate a new provisional date, proceeding in this cycle until a satisfactory fit is obtained.

Each actual computation of a planetary position has been a fairly laborious process. A book by P. V. Neugebauer, later slightly amended by some improved coefficients, organized the procedures into a systematic set of tables. While these tables were a real advance they still left a large amount of calculation to be done in each dating problem, and the computation of a single position might still take about twenty minutes for a person acquainted with the process.

There are a large number of astronomical records and tables surviving from the region of Babylon in the period from -600-0. Professors O. Neugebauer and A. Sachs, faced with the problem of dating these records, proposed to Dr. Goldstine of the Electronic Computer Project the production of an Ephemeris -- that is, a table of planetary, lunar and solar positions at regular intervals -- for the period in question. With the ephemeris

available, the dating of a record would be reduced to lacating the desired configuration in the ephemeris by inspection.

This problem was thought appropriate for the Electronic Computer Project for several reasons. Primarily, it was an example of a novel and useful application; while the amount of computation involved was of such magnitude as to be completely impractical without an electronic computer, it was of considerable but not excessive size on our computer. Secondly, it presented a number of interesting problems in high-precision calculations, error analysis, information handling, and methods of checking. Finally, the results were of a non-commercial nature and would be of permanent scientific value.

A thorough analysis of the problem was made from original sources. It was decided to use five-day intervals for Mercury, Venus, and the Moon, and ten-day intervals for Mars, Jupiter, Saturn, and the Sun. Except for the Moon the latitudes and longitudes are given in units of $.01^{\circ}$, and the intervals chosen are fine enough to permit interpolation with adequate accuracy by means of Everett's formula (for which a simplified method of use will be furnished). The positions of the Moon, whose motion is too rapid to permit satisfactory interpolation if a reasonable time interval is chosen, will be given in units of $.1^{\circ}$.

Work on this problem was commenced in the summer of 1955. It has inspired and been paralleled by the development of a number of coding aids, such as an assembly routine, a system of computer operation using "control-cards", and an assortment of utility sub-routines (see Part I). The problem is expected to enter production in the summer of 1957.

It is anticipated that the ephemeris, when produced and thoroughly checked, will be published by a scientific society as a monograph of about 300 pages of tables with accompanying text explaining the method of computation. To complement this publication, a Technical Report may be prepared for Contract Nonr-1358-(03), by which all mathematical analysis, programming and coding has been supported.

ADDENDUM:

The following two Technical Reports were issued under Contract No DA-36-034-ORD-1646:

Technical Report No 56 - 01, January 1956:

"A METHOD FOR FINDING THE GENERAL SOLUTION TO AN ARBITRARY NON-SINGULAR SYSTEM OF LINEAR EQUATIONS INVOLVING $n^3/2$ MULTIPLICATIONS"

by

J. Paul Roth

and

Technical Report No. 56 - 02, April 1956

"ALGEBRAIC TOPOLOGICAL METHODS FOR THE SYNTHESIS OF SWITCHING CIRCUITS IN n VARIABLES"

by

J. Paul Roth

This work is not included in our Final Report since full disclosure of the results was made in the Technical Reports and no machine operation was involved.